# Democratizing Information Access through Low Overhead Systems

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Benjamin Hättasch

Zur Erlangung des Grades eines
*Doctor rerum naturalium (Dr. rer. nat.)*
genehmigte Dissertation.

Referenten:
Prof. Dr. rer. nat. Carsten Binnig
Prof. Dr.-Ing. Sebastian Michel (RPTU Kaiserslautern-Landau)

Fachbereich Informatik
Technische Universität Darmstadt

Darmstadt, 2023

Democratizing Information Access through Low Overhead Systems

Accepted doctoral thesis by Benjamin Hättasch

Date of submission: 30.10.2023
Date of thesis defense: 11.12.2023

Darmstadt, Technische Universität Darmstadt

To everybody who supported me on my journey.

# Erklärungen laut Promotionsordnung

## § 8 Abs. 1 lit. c PromO

Ich versichere hiermit, dass die elektronische Version meiner Dissertation mit der schriftlichen Version übereinstimmt.

## § 8 Abs. 1 lit. d PromO

Ich versichere hiermit, dass zu einem vorherigen Zeitpunkt noch keine Promotion versucht wurde. In diesem Fall sind nähere Angaben über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs mitzuteilen.

## § 9 Abs. 1 PromO

Ich versichere hiermit, dass die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde.

## § 9 Abs. 2 PromO

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, 30.10.2023

B. Hättasch

# Abstract

Despite its importance, accessing information in storage systems or raw data is challenging or impossible for most people due to the sheer amount and heterogeneity of data as well as the overheads and complexities of existing systems. In this thesis, we propose several approaches to improve on that and therefore democratize information access.

Data-driven and AI based approaches make it possible to provide the necessary information access for many tasks at scale. Unfortunately, most existing approaches can only be built and used by IT experts and data scientists, yet the current demand for data scientists cannot be met by far. Furthermore, their application is expensive. To counter this, approaches with low overhead, i.e., without the need for large amounts of training data, manually annotating or extracting information, and extensive computation are needed. However, such systems still need to adapt to special terminology of different domains, and the individual information needs of the users. Moreover, they should be usable without extensive training; we thus aim to create ready-to-use systems that provide intuitive or familiar ways for interaction, e.g., chatbot-like natural language input or graphical user interfaces.

In this thesis, we propose a number of contributions to three important subfields of data exploration and processing: *Natural Language Interfaces for Data Access & Manipulation*, *Personalized Summarizations of Text Collections*, and *Information Extraction & Integration*. These approaches allow data scientists, domain experts and end users to access and manipulate information in a quick and easy way.

First, we propose two natural language interfaces for data access and manipulation. Natural language is a useful alternative interface for relational databases, since it allows users to formulate complex questions without requiring knowledge of SQL. We propose an approach based on weak supervision that augments existing deep learning techniques in order to improve the performance of models for natural language to SQL translation.
Moreover, we apply the idea to build a training pipeline for conversational agents (i.e., chatbot-like systems allowing to interact with a database and perform actions like ticket booking). The pipeline uses weak supervision to generate the training data automatically from a relational database and its set of defined transactions. Our approach is data-aware, i.e., it leverages the data characteristics of the DB at runtime to optimize the dialogue flow and reduce necessary interactions.
Additionally, we complement this research by presenting a meta-study on the reproducibility and availability of natural language interfaces for databases (NLIDBs) for real-world applications, and a benchmark to evaluate the linguistic robustness of NLIDBs.

Second, we work on personalized summarization and its usage for data exploration. The central idea is to produce summaries that exactly cover the current information need of the users. By creating multiple summaries or shifting the focus during the interactive creation process, these summaries can be used to explore the contents of unknown text collections. We propose an

approach to create such personalized summaries at interactive speed; this is achieved by carefully sampling from the inputs.
As part of our research on multi-document summary, we noticed that there is a lack of diverse evaluation corpora for this task. We therefore present a framework that can be used to automatically create new summarization corpora, and apply and validate it.

Third, we provide ways to democratize information extraction and integration. This becomes relevant when data is scattered across different sources and there is no tabular representation that already contains all information needed. Therefore, it might be necessary to integrate different structured sources, or to even extract the required information pieces from text collections first and then to organize them. To integrate existing structured data sources, we present and evaluate a novel end-to-end approach for schema matching based on neural embeddings.
Finally, we tackle the automatic creation of tables from text for situations where no suitable structured source to answer an information need is available. Our proposed approach can execute SQL-like queries on text collections in an ad-hoc manner, both to directly extract facts from text documents, and to produce aggregated tables stating information that is not explicitly mentioned in the documents. Our approach works by generalizing user feedback and therefore does not need domain-specific resources for the domain adaption. It runs at interactive speed even on commodity hardware.

Overall, our approaches can provide a quality level compared to state-of-the-art approaches, but often at a fraction of the associated costs. In other fields like the table extractions, we even provide functionality that is—to our knowledge—not covered by any generic tooling available to end users. There are still many interesting challenges to solve, and the recent rise of large language models has shifted what seems possible with regard to dealing with human language once more. Yet, we hope that our contributions provide a useful step towards democratization of information access.

# Zusammenfassung

Trotz ihrer Bedeutung ist der Zugang zu Informationen in Speichersystemen oder Rohdaten für die meisten Menschen aufgrund der schieren Menge und Heterogenität der Daten, sowie des Overheads und der Komplexität der bestehenden Systeme schwierig oder unmöglich. In dieser Arbeit schlagen wir mehrere Ansätze zur Verbesserung dieser Situation und damit zur Demokratisierung des Informationszugangs vor.

Datengetriebene und KI-basierte Ansätze machen es möglich, den notwendigen Informationszugang für viele Aufgaben skalierbar zur Verfügung zu stellen. Leider können die meisten existierenden Ansätze nur von IT-Expert*innen und *Data Scientists* erstellt und genutzt werden, wobei es jedoch längst nicht genug *Data Scientists* gibt. Zudem ist der Einsatz teuer. Um dem abzuhelfen, werden Ansätze mit geringem Overhead benötigt, d. h. ohne die Notwendigkeit großer Mengen von Trainingsdaten, manueller Annotation oder Extraktion von Informationen und umfangreicher Berechnungen. Solche Systeme müssen sich jedoch an die spezielle Terminologie verschiedener Disziplinen und den individuellen Informationsbedarf der Nutzer anpassen. Darüber hinaus sollten sie ohne umfangreiches Training nutzbar sein; wir wollen daher direkt nutzbare Systeme schaffen, die intuitive oder vertraute Interaktionsmöglichkeiten bieten, z. B. Chatbot-ähnliche natürlichsprachliche Eingaben oder grafische Benutzeroberflächen.

In dieser Arbeit schlagen wir eine Reihe von Beiträgen zu drei wichtigen Teilbereichen der Datenexploration und -verarbeitung vor: Natürlichsprachliche Schnittstellen für Datenzugriff und -manipulation, personalisierte Zusammenfassungen von Textsammlungen und Informationsextraktion und -integration. Diese Ansätze ermöglichen es *Data Scientists*, Fachleuten und Endanwender*innen, schnell und einfach auf Informationen zuzugreifen und sie zu bearbeiten.

Als Erstes schlagen wir zwei natürlichsprachliche Schnittstellen für den Datenzugriff und die Datenmanipulation vor. Natürliche Sprache ist eine nützliche alternative Schnittstelle für relationale Datenbanken, da sie es den Anwender*innen ermöglicht, komplexe Fragen zu formulieren, ohne dass sie Kenntnisse über SQL benötigen. Wir schlagen einen auf *weak supervision* basierenden Ansatz vor, der bestehende *Deep-Learning*-Techniken augmentiert, um die Leistung von Modellen für die Übersetzung von natürlicher Sprache zu SQL zu verbessern.
Außerdem wenden wir die Idee an, um eine Trainingspipeline für conversational agents (d.h. Chatbot-ähnliche Systeme, die es ermöglichen, mit einer Datenbank zu interagieren und Aktionen wie Ticketbuchungen durchzuführen) zu konstruieren. Die Pipeline nutzt *weak supervision*, um die Trainingsdaten automatisch aus einer relationalen Datenbank und einer Reihe von definierten Transaktionen zu generieren. Unser Ansatz ist *data-aware*, d.h. er nutzt die Dateneigenschaften der DB zur Laufzeit, um den Dialogfluss zu optimieren und die notwendigen Interaktionen zu reduzieren.
Darüber hinaus ergänzen wir diese Forschung, indem wir eine Metastudie über die Reproduzierbarkeit und Verfügbarkeit von natürlichsprachlichen Interfaces für Datenbanken (NLIDBs) für reale

Anwendungen sowie einen Benchmark zur Bewertung der linguistischen Robustheit von NLIDBs vorstellen.

Als Zweites befassen wir uns mit personalisierten Zusammenfassungen und deren Nutzung für die Datenexploration. Die zentrale Idee ist es, Zusammenfassungen zu erstellen, die genau den aktuellen Informationsbedarf der Nutzenden abdecken. Durch die Generierung mehrerer Zusammenfassungen oder die Verschiebung des Fokus während des interaktiven Erstellungsprozesses können diese Zusammenfassungen zur Exploration des Inhalts unbekannter Textsammlungen verwendet werden. Wir schlagen einen Ansatz vor, um solche personalisierten Zusammenfassungen in interaktiver Geschwindigkeit zu erstellen; dies wird durch sorgfältiges Sampeln aus den Eingaben erreicht.
Im Rahmen unserer Forschung zu Multi-Dokument-Zusammenfassung haben wir einen Mangel an diversen Evaluierungskorpora für diese Aufgabe ausgemacht. Wir stellen daher ein Framework vor, das zur automatischen Erstellung neuer Zusammenfassungskorpora verwendet werden kann, wenden es an und validieren es.

Als Drittes stellen wir Möglichkeiten zur Demokratisierung der Informationsextraktion und -integration vor. Dies wird dann relevant, wenn die Daten über verschiedene Quellen verstreut sind und es keine tabellarische Darstellung gibt, die bereits alle benötigten Informationen enthält. Daher kann es notwendig sein, verschiedene strukturierte Quellen zu integrieren oder sogar die benötigten Informationen zunächst aus Textsammlungen zu extrahieren und sie dann zu organisieren. Um bestehende strukturierte Datenquellen zu integrieren, präsentieren und evaluieren wir einen neuartigen Ende-zu-Ende-Ansatz für *Schema-Matching*, der auf neuronalen *Embeddings* basiert.
Abschließend befassen wir uns mit der automatischen Erstellung von Tabellen aus Text für Situationen, in denen keine geeignete strukturierte Quelle zur Beantwortung eines Informationsbedarfs verfügbar ist. Der von uns vorgeschlagene Ansatz kann SQL-ähnliche Abfragen auf Textsammlungen ad hoc ausführen – sowohl um Fakten direkt aus Textdokumenten zu extrahieren, als auch um durch Filterung, Aggregation und Gruppierung Tabellen mit Informationen zu erstellen, die nicht explizit in den Dokumenten erwähnt werden. Unser Ansatz basiert auf der Generalisierung von Benutzerfeedback und benötigt daher keine domänenspezifischen Ressourcen für die Anpassung an die jeweilige Fachsprache. Er läuft in interaktiver Geschwindigkeit sogar auf Standard-Hardware.

Zusammengefasst können unsere Ansätze ein Qualitätsniveau bieten, das mit *State-of-the-art*-Ansätzen vergleichbar ist, aber das in vielen Fällen zu einem Bruchteil der damit verbundenen Kosten. In anderen Bereichen, wie z.B. bei der Extraktion von Tabellen, bieten wir sogar Funktionen, die – unseres Wissens nach – bisher von keinem generischen Tool für Endbenutzer*innen bereitgestellt werden. Es verbleiben noch viele interessante Herausforderungen und die kürzlichen Fortschritte bei großen Sprachmodellen (LLMs) hat die Grenzen des Möglichen im Umgang mit menschlicher Sprache noch einmal verschoben. Dennoch hoffen wir, dass unsere Beiträge einen nützlichen Schritt zur Demokratisierung des Informationszugangs darstellen.

# Publications

The following peer-reviewed publications are part of this cumulative dissertation. Their content is printed in Part II, Chapters 7 to 18.


[Bas+18] Fuat Basik, Benjamin Hättasch, Amir Ilkhechi, Arif Usta, Shekar Ramaswamy, Prasetya Utama, Nathaniel Weir, Carsten Binnig, and Ugur Çetintemel. "DBPal: A Learned NL-Interface for Databases". In: *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*. Edited by Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein. ACM, 2018, pages 1765–1768. DOI: 10.1145/3183713.3193562.

[Hät18] Benjamin Hättasch. "Towards Interactive Summarization of Large Document Collections". In: *Proceedings of the First Biennial Conference on Design of Experimental Search & Information Retrieval Systems, Bertinoro, Italy, August 28-31, 2018*. Edited by Omar Alonso and Gianmaria Silvello. Volume 2167. CEUR Workshop Proceedings. CEUR-WS.org, 2018, page 103. URL: https://ceur-ws.org/Vol-2167/short6.pdf.

[HMB19] Benjamin Hättasch, Christian M. Meyer, and Carsten Binnig. "Interactive Summarization of Large Document Collections". In: *Proceedings of the Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD 2019, Amsterdam, The Netherlands, July 5, 2019*. ACM, 2019, 9:1–9:4. DOI: 10.1145/3328519.3329129.

[Hät+20a] Benjamin Hättasch, Nadja Geisler, Christian M. Meyer, and Carsten Binnig. "Summarization Beyond News: The Automatically Acquired Fandom Corpora". In: *Proceedings of The 12th Language Resources and Evaluation Conference, LREC 2020, Marseille, France, May 11-16, 2020*. Edited by Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asunción Moreno, Jan Odijk, and Stelios Piperidis. European Language Resources Association, 2020, pages 6700–6708. URL: https://aclanthology.org/2020.lrec-1.827/.

[Hät+20b] Benjamin Hättasch, Michael Truong-Ngoc, Andreas Schmidt, and Carsten Binnig. "It's AI Match: A Two-Step Approach for Schema Matching Using Embeddings". In: *2nd International Workshop on Applied AI for Database Systems and Applications (AIDB20). In conjunction with the 46th International Conference on Very Large Data Bases, Virtual, August 31 - September 4, 2020*. Edited by Bingsheng He, Berthold Reinwald, and Yingjun Wu. Virtual, 2020. DOI: 10.48550/arXiv.2203.04366.

[Wei+20] Nathaniel Weir, Prasetya Utama, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Nadja Geisler, Benjamin Hättasch, Steffen Eger, et al. "DBPal: A Fully Pluggable NL2SQL Training Pipeline". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. Edited by David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo. ACM, 2020, pages 2347–2361. DOI: 10.1145/3318464.3380589.

[Hät21] Benjamin Hättasch. "WannaDB: Ad-hoc Structured Exploration of Text Collections Using Queries". In: *Proceedings of the Second International Conference on Design of Experimental Search & Information REtrieval Systems, Padova, Italy, September 15-18, 2021*. Edited by Omar Alonso, Stefano Marchesin, Marc Najork, and Gianmaria Silvello. Volume 2950. CEUR Workshop Proceedings. CEUR-WS.org, 2021, pages 179–180. URL: https://ceur-ws.org/Vol-2950/paper-23.pdf.

[HBB21] Benjamin Hättasch, Jan-Micha Bodensohn, and Carsten Binnig. "ASET: Ad-hoc Structured Exploration of Text Collections". en. In: *3rd International Workshop on Applied AI for Database Systems and Applications (AIDB21). In conjunction with the 47th International Conference on Very Large Data Bases, Copenhagen, Denmark, August 16 - 20, 2021*. Copenhagen, Denmark, 2021. arXiv: 2203.04663.

[HGB21] Benjamin Hättasch, Nadja Geisler, and Carsten Binnig. "Netted?! How to Improve the Usefulness of Spider & Co". In: *Proceedings of the Second International Conference on Design of Experimental Search & Information REtrieval Systems, Padova, Italy, September 15-18, 2021*. Edited by Omar Alonso, Stefano Marchesin, Marc Najork, and Gianmaria Silvello. Volume 2950. CEUR Workshop Proceedings. CEUR-WS.org, 2021, pages 38–43. URL: https://ceur-ws.org/Vol-2950/paper-08.pdf.

[Gas+22] Marius Gassen, Benjamin Hättasch, Benjamin Hilprecht, Nadja Geisler, Alexander Fraser, and Carsten Binnig. "Demonstrating CAT: Synthesizing Data-Aware Conversational Agents for Transactional Databases". In: *Proc. VLDB Endow.* 15.12 (2022), pages 3586–3589. URL: https://www.vldb.org/pvldb/vol15/p3586-h%5C%e4ttasch.pdf.

[HBB22] Benjamin Hättasch, Jan-Micha Bodensohn, and Carsten Binnig. "Demonstrating ASET: Ad-hoc Structured Exploration of Text Collections". In: *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. ACM, 2022, pages 2393–2396. DOI: 10.1145/3514221.3520174.

[Hät+23a] Benjamin Hättasch, Jan-Micha Bodensohn, Liane Vogel, Matthias Urban, and Carsten Binnig. "WannaDB: Ad-hoc SQL Queries over Text Collections". In: *Datenbanksysteme für Business, Technologie und Web (BTW 2023), 20. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 06.-10, März 2023, Dresden, Germany, Proceedings*. Edited by Birgitta König-Ries, Stefanie Scherzinger, Wolfgang Lehner, and Gottfried Vossen. Volume P-331. LNI. Gesellschaft für Informatik e.V., 2023, pages 157–181. DOI: 10.18420/BTW2023-08.

Additionally, the following publication which is under submission is printed in Chapter 19.

[Hät+23b] Benjamin Hättasch, Liane Vogel, Gard Jenset, Jan-Micha Bodensohn, Chandrima Roy, and Carsten Binnig. "WannaDB in Action: Deploying Ad-hoc SQL-over-Text Exploration in an Industrial Scenario". In: *Under submission* (2023).

Further co-authored peer-reviewed publications are:

[Avi+18] Avinesh P. V. S., Benjamin Hättasch, Orkan Özyurt, Carsten Binnig, and Christian M. Meyer. "Sherlock: A System for Interactive Summarization of Large Text Collections". In: *Proc. VLDB Endow.* 11.12 (2018), pages 1902–1905. ISSN: 2150-8097. DOI: 10.14778/3229863.3236220.

[Hil+20] Benjamin Hilprecht, Carsten Binnig, Tiemo Bang, Muhammad El-Hindi, Benjamin Hättasch, Aditya Khanna, Robin Rehrmann, Uwe Röhm, Andreas Schmidt, Lasse Thostrup, and Tobias Ziegler. "DBMS Fitting: Why should we learn what we already know?" In: *10th Conference on Innovative Data Systems Research, CIDR 2020, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings*. www.cidrdb.org, 2020. URL: http://cidrdb.org/cidr2020/papers/p34-hilprecht-cidr20.pdf.

[GHB22] Nadja Geisler, Benjamin Hättasch, and Carsten Binnig. "Demonstrating Quest: A Query-Driven Framework to Explain Classification Models on Tabular Data". In: *Proc. VLDB Endow.* 15.12 (2022), pages 3722–3725. URL: https://www.vldb.org/pvldb/vol15/p3722-geisler.pdf.

[HB22] Benjamin Hättasch and Carsten Binnig. "Know Better - A Clickbait Resolving Challenge". In: *Proceedings of the Thirteenth Language Resources and Evaluation Conference, LREC 2022, Marseille, France, 20-25 June 2022*. Edited by Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Jan Odijk, and Stelios Piperidis. European Language Resources Association, 2022, pages 515–523. URL: https://aclanthology.org/2022.lrec-1.54.

Additionally, I am co-author of the following preprints:

[Uta+18] Prasetya Utama, Nathaniel Weir, Fuat Basik, Carsten Binnig, Ugur Cetintemel, Benjamin Hättasch, Amir Ilkhechi, Shekar Ramaswamy, and Arif Usta. *An End-to-end Neural Natural Language Interface for Databases*. 2018. arXiv: 1804.00401 [cs.DB].

[Wei+19b] Nathaniel Weir, Andrew Crotty, Alex Galakatos, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Ugur Cetintemel, Prasetya Utama, Nadja Geisler, Benjamin Hättasch, Steffen Eger, and Carsten Binnig. *DBPal: Weak Supervision for Learning a Natural Language Interface to Databases*. 2019. arXiv: 1909.06182 [cs.DB].

Due to the nature of the synopsis and for better readability, selected paragraphs from these publications were transferred verbatim throughout the synopsis without explicit labeling, as suggested in the department regulations "Kumulative Dissertation und Eigenzitate in Dissertationen" (21.09.2021) §1.

# Contents

# Part I.

# Synopsis

# 1. Introduction

## 1.1. The Need for Democratization

Getting (the right) information from large amounts of data is important in many fields, from research over healthcare and public service to journalism: Researchers have to find relevant existing work. City administrations want to learn about the needs of their citizens from large numbers of complaints. Fiscal authorities try to uncover tax evasion and money laundering. Journalists need facts and statistics to back their articles. And healthcare professionals interpret test results and patients files to learn about their patients and how to help them.

Computers with their ability to quickly store and process enormous amounts of data, and the progresses in artificial intelligence (AI) (e.g., pattern detection, automatic translation, language modelling and much more) offer great new opportunities for that. Data-driven AI methods and learned models are therefore getting more and more important to solve these issues at scale [e.g., Bar+15; Mor+19; Sko+19]. They can help to take decisions (e.g., which material to use, which articles to believe in, which paper to read, or which company to check more thoroughly) in an informed and therefore advantageous way.

Unfortunately, those approaches are often associated with high effort and overhead, and can only be used by AI experts. A wide application, however, can only be reached by a democratization that makes these approaches usable for more people. That importance is underlined by the growing demand for data scientists that currently cannot be met by far [Bur23; DP22; Wor23], slowing down scientific, industrial and societal development and progress.

One central reason for this high demand lies in the high manual effort in finding and preparing the right data. Different studies say that data scientists spend up to $80\,\%$ of their time on preparation tasks like data cleaning, organizing and annotating it manually [Ana21; Pre16]. Being able to reduce this amount, e.g., through approaches that require less annotated data, or automatically extract relevant information from raw data, would allow them to concentrate more on building models, analyzing the data itself, and creating value.

Another reason preventing the wide application of these approaches are the overheads of the approaches itself, e.g., for training them, running predictions on thousands of documents, or waiting for the results. This makes analysis and exploration of new data expensive—or might completely prevent it, if the users do not have access to the necessary computing power, but cannot upload their data, e.g., due to legal or privacy reasons, to an external provider either. *Smaller models* and *approximation* can help here to get the necessary answers faster and at much lower costs.

Finally, even with reduced efforts and costs to build and run AI-based approaches, there will most likely not be enough experts available such that every process, company and institution can profit

from the advantages. One can only expect a wide adaptation if these tools can directly be used by those people currently doing the job manually (e.g., handling insurance claims). Therefore, these systems need to guide the users and require as little of complicated actions from them as possible—while that level of automatization has to be reached with low overhead again.

To summarize, *systems with low overhead and complexity* (both with regard to ease of use and computation) can lead to a *democratization* that allows (data) scientists, domain experts and end users to access information, and explore and analyze data without special knowledge, skills and complicated training. In this thesis, we present contributions to tackle these directions, as we will discuss below.

## 1.2. Challenges

This thesis aims to show how the rapid advances in artificial intelligence, more precisely in the field of text and information processing, can be applied in a way that benefits as many as possible. To do so, and make information that is present "somewhere" really available to a user, this thesis tries to overcome a series of challenges:

**Dealing with large amounts of data:** In many cases, large amounts of data, e.g., text documents, need to be processed to distill the information a user needs. Yet, users can often only read or manually process a small fraction of them in a reasonable time. As a result, they have to work with incomplete or suboptimal information. Moreover, for exploratory use-cases, the time spent processing irrelevant sources has to be considered as "wasted", limiting the possible scope of explorations. A *good* system should therefore automatize time-consuming steps, reducing the overall time to explore data and access information drastically.

**Interactive exploration of data is needed:** To explore data and get an overview of it, users need both useful and quick responses. As an example, many people will more probably issue a new adapted search query than looking at the second page of Google search results when the first ten results out of millions of possibilities do not match their expectations. Systems need to produce short and concise results with little information. Since the computer cannot magically "know" what they mean exactly, interaction and multiple tries have to be possible. Therefore, results have to be produced in a short timespan, preferably without the need for special or powerful hardware. Thus, one needs to find ways to guarantee low processing and interaction times, preferably on commodity hardware.

**Dealing with domain-specific language & individual information needs:** Human language is highly ambiguous, context-depending, subjective and, in particular, domain-specific. This can already be seen inside a field (e.g., the word *collection* has different meanings in software engineering and natural language processing) and is even stronger across disciplines. Yet, domain-specific words like medical terminology are often important and semantically bearing for the tasks applied to the data, thus, adapting a system to them is crucial. A particular challenge is treating numbers correctly, since they often offer even less hints on their semantics than words do. Systems for information access and exploration will only be of use when they can deal with these properties

of language; they need to bridge between the user's wording and the terms of existing resources such that, e.g., technical terms are correctly extracted from input documents, or that the user can interact with them using their domain-specific language. Involving domain experts for this adaption is costly and often not feasible. AI approaches can resolve some of the challenges mentioned before by automatically processing data and, e.g., applying text extraction—but this again requires some kind of adaption to the specific use case. Additionally, users might expect a different granularity of details; things that are considered as the same for one person might need to be treated as completely different for another. Thus, not only the correct handling of input, but also a customization of outputs is needed. In short, it would be very hard to build a "one fits all" solution. To accomplish all this adjustment, AI approaches often require annotated data for training or fine-tuning, which unfortunately might be difficult and expensive to acquire or might not be available at all. A *good* system nevertheless needs to adapt to the user and provide customized results—without introducing a big overhead to do so.

**Heterogeneity of data:** The data relevant for an information need might be in an unusable format (e.g., running text instead of structured tables) for the task at hand, preventing directly computing or compiling the result representation needed. Therefore, automatic approaches to transfer them from one format into another might be required. Relevant information might be spread across sources (like different databases or text files) and those sources might follow different naming conventions (e.g., schemata of various tables in a data lake). Hence, implicitly or explicitly unifying these information sources is another challenge. Often, the schema or underlying structure might be completely unknown to those using it, be it with data lakes for companies or in exploratory scenarios for end users. Hence, approaches to organize and explore large amounts of heterogeneous data are needed.

**Overwhelmed users:** Programming is not a skill that basically everybody is capable of; even computer science students after their first year of study struggle with holding viable mental models of the tasks and what they are doing [Ma+07]. A more humorous approach to show how difficult it can be to give precise instructions that really lead to the expected result, is asking people to write down the instructions to make a peanut butter and jelly sandwich and show them what happens when one follows the instructions without using additional knowledge.[1] Thus, to democratize information access, systems need to work without programming or specifying rules, regular expressions, or complex queries. Instead of requiring the user to guide the computer, computers have to guide the users, since even with ready-to-use systems, many users are overwhelmed, both with the question where to find the relevant information and then how to actually access it. Therefore, one has to create systems that are intuitively to use and only require a minimum of learning. The more complex a system is, the more training or instructions its usage needs, the more barriers are introduced and the less it will be adapted by (domain) users.

**Overhead and other downsides of existing approaches:** Many learned approaches try to solve both problems that are hard to formalize (e.g., semantic understanding), and those that would profit from mathematic rigorousness (e.g., aggregation and result computation), end-to-end, hence one cannot expect even simple summing operations to be correct all the time when depending on

[1] See, e.g., this video: `https://www.youtube.com/watch?v=cDA3_5982h8`

such components. This in particular holds for large language models (LLMs) which are currently applied to a lot of research areas. They often provide astonishing levels of natural language understanding (NLU), but are expensive to use (high computation or per-query costs). Enforcing a domain-adaptation (e.g., through carefully crafted prompts and examples) can be difficult. Often, parts of the results are not based on the inputs but derived from background knowledge, but there is a lack of transparency with regard to what answers are based on, and updating the background knowledge (by retraining) is highly expensive. Finally, it can be hard to detect when good-looking but wrong results are produced [Ban+23].

## 1.3. Problems & Solutions

In this thesis, we will show how we overcame these challenges by several contributions to the fields of *Natural Language Interfaces for Data Access & Manipulation*, *Personalized Summarizations of Text Collections*, and *Information Extraction & Integration*.

In summary, the methods proposed in this thesis tackle two central facets: First, they try to ensure that it becomes possible to use automatic processing of data in certain or arbitrary domains at all. Second, the approaches should put users at the center, be accessible to them and tailored to their needs. To accomplish this, in this thesis we will propose a series of new approaches united by the fact that they make the application of the novel technologies possible in the first place through low effort and low cost—and at the same time try to keep the barriers to actually use them as low as possible.

It is probably not surprising that the challenges mentioned above cannot be solved for every kind of information and data at once. We will thus focus on one kind of information and query representation. While formalization, e.g., through mathematical expressions, play a central role for some subfields of knowledge representation, in many other areas text or words are used extensively. In this thesis, we will therefore concentrate on several fields that are related to human language. Such textual knowledge can both exist in structured form (e.g., tables and databases), and highly unstructured (as running text). For both of these kinds, in this thesis we will present approaches to address the challenges mentioned above.

We thus need to present ways to deal with large amounts of data, in forms of big databases and data lakes, and as written text. The systems we propose need to provide the users with ways to mentally handle the amount of diverse information contained in that data. Additionally, they need to scale well from a computational perspective, to deliver the answers to information needs in time—often in only minutes or even seconds. Our approaches should work out of the box on domains not seen or known before. They need to correctly deal with domain-specific terminology (e.g., column names or entities to be extracted) and should adapt to the users' formulations and not vice versa. All systems proposed in this thesis will have a low overhead, reaching this domain adaption without the need to involve domain experts and by requiring substantially lower amounts of feedback or annotations than most state-of-the-art systems. While this thesis is not a one in the area of human computer interaction, we will nevertheless put a focus on building systems that can be used by end users, e.g., through providing graphical or natural language interfaces. Finally, we will explore how existing resources like language models (LMs) can be leveraged to profit from their rapid advancements for, e.g., NLU without introducing the downsides of many

current end-to-end natural language processing (NLP) approaches (e.g., hallucination, prediction instead of computation for mathematical tasks).

We will now quickly show how we contribute to democratization in the different fields mentioned before. For the details, we refer to the following chapters.

**Natural Language Interfaces for Data Access & Manipulation:** To access structured data or to even manipulate it, one usually needs knowledge about the structure of the data and must be proficient in a query language like SQL or SPARQL. Being able to instead interact with the data using well-known human language would lower the barrier drastically. Natural language interfaces for databases (NLIDBs) and conversational agents (CAs) allow this. However, the construction of such systems often involves a high degree of effort and expenses, since example data needs to be collected and manually annotated. We therefore propose approaches to automatically generate such training data from considerably smaller amounts of input data (*weak supervision*) in Chapter 3. We show how this idea can be used to generate both NLIDBs (*DBPal*, see Section 3.1) and CAs (*CAT*, see Section 3.3) with drastically reduced effort and cost, and therefore allow for a wide application. At the same time, we analyze how the computer can lead users in their interaction with the data such that they can perform this interaction without knowledge about the structure of the data. In addition to own approaches, we evaluate the progress of the community to build real-world systems for that field (see Section 3.2).

**Personalized Summarizations of Text Collections:** While tabular data at least has a structure (even if the user might not know it), for running texts often neither the users nor the computer have any information about the underlying structure and relations. Getting such an overview manually comes at high costs/effort for large text collections (e.g., the Panama Papers). Thus, aid by the computer is desirable: automatically generated summaries can be helpful to drastically reduce the amounts of pages that have to be read to gain insights. Therefore, they can support experts like journalists, and in general make relevant contents directly accessible to a wider range of people. Yet, by their nature, summaries are lossy, only a fraction of the information contained in hundreds of pages can be condensed in a few lines of text. To make sure this subset then covers the useful information, i.e., the one that is relevant for the information need of the user, we propose an approach for generating personalized summarizations (*Sherlock*, see Section 4.1). Our approach works interactively and without domain or user specific data (which would be hard to gather). Additionally, we address how such approaches can be evaluated. As part of that, we propose and apply a framework for automatic acquisition of additional summarization corpora (*Fandom Corpora*, see Section 4.2).

**Information Extraction & Integration:** A textual summarization can be very useful to get an overview of a text collection and the information contained in it. Yet, in order to answer many kinds of questions or to assess hypotheses, a more structured kind of summary is needed—like a tabular representation. The third and last big area of our research therefore deals with providing such structured information to the users. As a central part to accomplish this, we present an approach where user and computer together fill a user-defined table from a suitable text collection in an interactive process (*WannaDB*, see Section 5.2). To do so, the computer requests feedback actions from the user and generalizes the information received to fill large parts of the table without

requiring the users to read the corresponding source texts. Furthermore, we present an approach to combine existing tables and databases with different schemata and even across languages (see Section 5.1). Again, the focus here is to make these approaches widely applicable both with regard to the associated costs and efforts as well as the necessary skills.

## 1.4. This Thesis

In the following chapter (Chapter 2), we will describe the design principles used, and outline the usage and the core ideas of our approaches, before covering the details of them in Chapters 3 to 5. Afterwards, a summary and an outlook follow in Chapter 6. The details and extensive evaluation results for all approaches discussed in this thesis can be found in Part II, Chapters 7 to 19, where the publications we created in the last years to advance research in these areas are included in this thesis.

# 2. Approach & Contributions

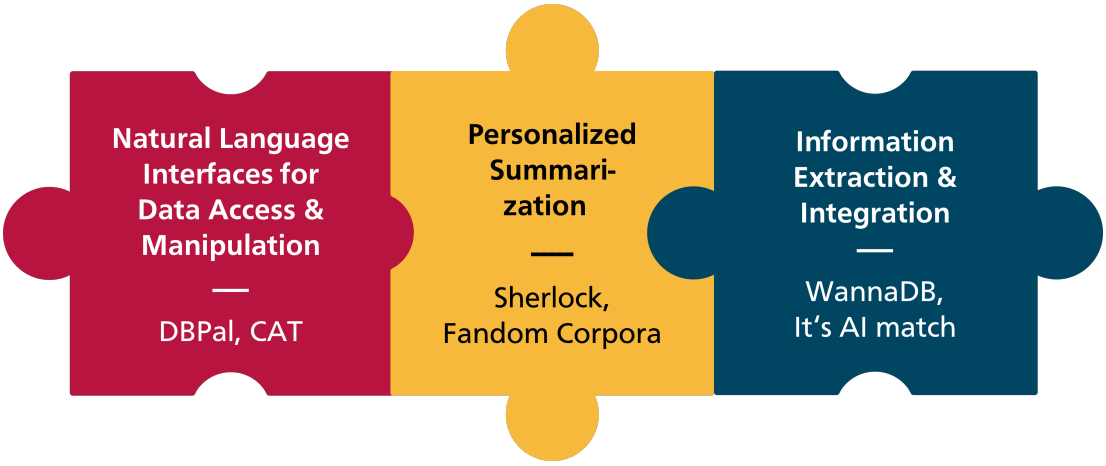



Figure 2.1.: Overview of the research directions and projects covered in this thesis.

In this chapter, we will give an overview of the directions towards democratization of information access we propose in this thesis. We will start with an exemplary use-case where all proposed research directions are used for exploring data and resolving an information need. Afterwards, we will spotlight the common principles used when building all of our approaches. The remaining structure of the chapter will follow the three directions previously introduced in Section 1.3, namely *Natural Language Interfaces for Data Access & Manipulation*, *Personalized Summarizations of Text Collections*, and *Information Extraction & Integration*. For each of these directions, we present one or more systems that allow information access through these means (see Figure 2.1). However, we argue that these approaches should not be seen in isolation, but can—when needed—be used as a full pipeline for data exploration, which we will outline in the last section of this chapter.

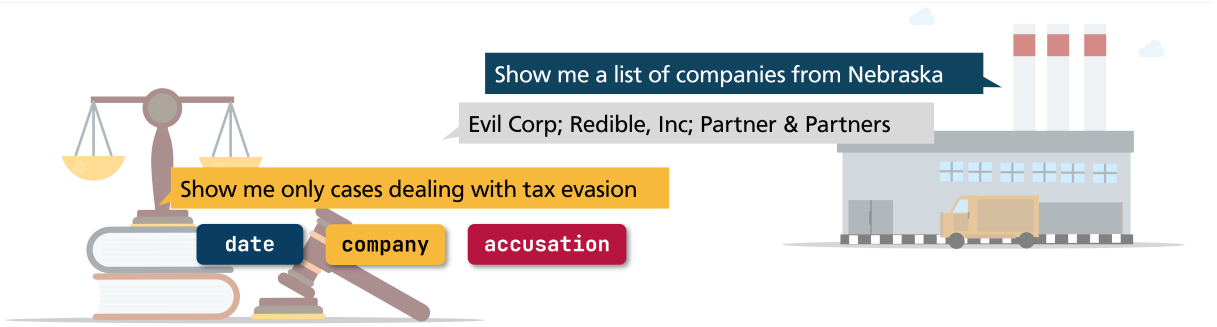
## 2.1. Application Example

In April 2016, over 100 media outlets from all over the world present their first results of analysis of the *Panama Papers*, a collection of 11.5 million text documents, containing information about more than 200,000 offshore companies, largely founded with the aim of tax evasion and hiding other kinds of frauds and crimes. These news publications are the result of a one-year long careful analysis by many trained data journalists, using special tools like *Nuix*.[1]

[1] `https://www.nuix.com/`

Imagine a journalist working for a smaller news outlet. Triggered by this event, they get the task to report on companies and people from their town or area that are or were involved in frauds. They probably do not have experience with professional tools to extract information from large text collections, they cannot program, and their company cannot afford to provide powerful computing infrastructure or pay high computing or license fees. Yet, their audience wants to know about the local impact of the scandal right away. Manually scrolling through databases and reading thousands of pages of text is not an option.

Our tool chain provides an alternative: The journalist might start by accessing existing structured data, like a database of registered companies, through a natural language interface (NLI) as we propose it with *DBPal* and *CAT* (see Chapter 3). Using natural language interfaces for the data access makes this possible for them if they do not know SQL. And even if they do, posing queries in natural language instead of needing to remember the correct syntax, column names and value representations will be much easier and faster for most users. The NLI helps to learn, explore a domain, and get structured results for questions. For some use-cases, that might already be enough, but in our example, the journalist needs more information that is not covered yet, since the data source does not elaborate on any wrong behavior in the past.



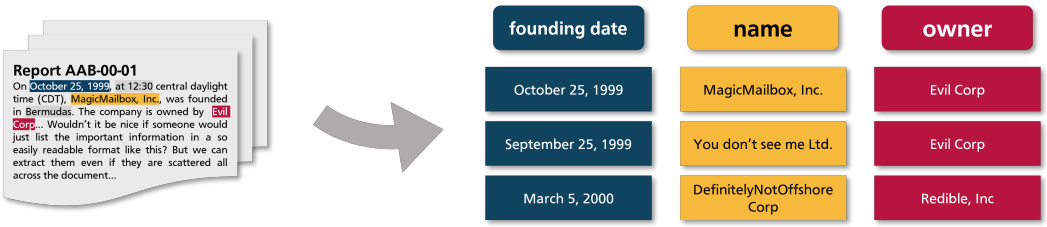

A first approach would therefore be unifying resources to combine data from multiple structured sources. This can help to get more data points for the same issue covered in multiple sources, or to even find new patterns and connections. The journalist may, e.g., link company information and a legal case database. To bridge between the different domain-specific wording in the two sources, embedding-based matching as we propose in Section 5.1 can be used.


A second approach to find additional connections and facts can be applied when no suitable structured data is available: find and exploit information in unstructured data (e.g., text collections). In our sample case, the journalist might access the public part of the *Panama Papers* or other collections disclosing similar information, like the *Offshore Leaks* from 2013.

This will often again be a multistep approach: First, the journalist needs to check whether the text collection contains relevant information (e.g., whether companies from the area are mentioned). In other use-cases, a user might not even know exactly what they are looking for: a journalist might have received a large collection of internal emails from a company by a whistleblower and is now openly exploring what might be covered in them, assuming they will contain information about some kind of misbehavior but not knowing the details of what that might be. In both cases

(targeted and open exploration), a system for personalized summarization as we propose it with *Sherlock* (see Chapter 4) can be very useful, as it provides the user with an overview of the contents of a document collection that are related to a certain aspect. Moreover, the interaction paradigm allows them to adjust that focus during the exploration based on the result received so far and does not require them to already exactly know what they are looking for in advance.

Once they have an overview of the unstructured data at hand, in the final step, the journalist might again be interested in more structured results to answer specific questions. In our example, they might have learned that indeed their city or state is mentioned in multiple places throughout the document collection and now want to compile a list of the specific companies involved, so they can check them more thoroughly. Our final contribution *WannaDB* (see Section 5.2) can help them with that. It allows specifying custom result table structures and have them filled in a mostly automatic process from source text documents, based on some rounds of interactive feedback to adjust for domain-specific language.




In our example, our journalist ends with a short table of companies, involved financial institutes or transaction details, created from the large document collection of offshore transaction leaks. They can now start writing their first article on that topic, or have at least a direction to continue their investigation. Hence, they might now grab their phone and contact an informant that might know more about an aspect they just learned about—but might come back to the data later to check more ideas, aspects, and hypotheses. The tools we propose helped them to gain these insides in a short timespan, with little costs for exploration and without any programming.

## 2.2. Design Principles

Before we go into detail on the different directions, we outline the design principles we followed for all of them to face the challenges described in Section 1.2.

To create systems that can be of use for many people, we apply four basic design principles throughout our research:

**Automatization:** Moving long, complex or repetitive tasks (e.g., extracting information or comparing entries) from the user to the computer both reduces the necessary skills and knowledge, and allows for scale-up.

**Usability:** Second, again to not depend on skills like programming or another strong computer or data science background, all systems should provide an interface that the user is already familiar with or can quickly learn—e.g., by the use of natural language or a graphical interface.

**(Cost) Efficency:** Third, we need to keep direct (e.g., license fee and computation bills) or indirect costs (e.g., caused by the time the user needs to spend) for information access low. We reach that by using open and comparatively small resources, by reducing the amount of annotated data that needs to be present, and, in particular, by approximation and sampling.

**Responsiveness:** Our systems can (thanks to the previous principle) be made fast. The last principle is to exploit this, such that the user can quickly interact with them, either to adjust their request, or directly as part of the process. Therefore, even with approximate systems, users can improve the result quality by pursuing further actions with the system—but, in contrast to other systems, they only need to do this until they reached a quality that matches *their* information need.

## 2.3. Natural Language Interfaces for Data Access & Manipulation

Our first research area are natural language interfaces for data access and manipulation. Natural language is a promising alternative interface to database management systems (DBMSs) because it enables non-technical users to formulate complex questions in a more concise manner than they could with SQL. In the last years, the number of deep learning approaches for translating natural language to SQL grew [KK23; Kim+20; Qin+22], and also the usage of foundational large language models (LLMs) for that task gained traction. Even with LLMs like *ChatGPT*, this problem is not magically solved, since they can advance the natural language understanding (NLU) component of the task, but they still require good mechanisms to generate suitable prompts [Gao+23], so one needs a trained system accompanying the LLM to reach high quality. Alternatively, the possible quality that can be reached with open source LLMs is much lower than the quality of state-of-the-art closed ones [Sun+23], but can be drastically improved with fine-tuning [Gao+23]. Hence, all of these approaches require an enormous amount of training data in order to provide accurate translations. This training data is extremely expensive to curate, since it generally requires humans to manually annotate natural language examples with the corresponding SQL queries (or vice versa).

Based on these observations, we propose a new approach that augments existing deep learning techniques in order to improve the performance of models for natural language to SQL (NL2SQL) translation (see Section 3.1). More specifically, we present a novel training pipeline that automatically generates synthetic training data in order to (1) improve overall translation accuracy, (2) increase robustness to linguistic variation, and (3) specialize the model for the target database. As we show, our training pipeline is able to improve both the accuracy and linguistic robustness of state-of-the-Art NL2SQL translation models.

As part of the research on this topic, we also developed and published a benchmark called *ParaphraseBench* (see Section 3.1.3), that tests how well an NL2SQL approach can deal with linguistic variations of the same query. Our evaluations show that the quality of the translation can vary drastically for different natural language (NL) formulations of the same SQL query, and hint which kinds of variations models generalize rather good or bad, too.

**Contributions:** We present *DBPal*, a fully pluggable natural language to SQL (NL2SQL) training pipeline that automatically synthesizes training data in order to improve the translation accuracy of an existing deep learning-based model. We propose several data augmentation techniques that give the model better coverage and make it more robust towards linguistic variation in NL queries. We propose a new benchmark that systematically tests the robustness of a natural language interface for databases (NLIDB) to different linguistic variations. Using a state-of-the-art deep learning model, we show that our training pipeline can improve translation accuracy by up to almost $40\,\%$.

To complement this research, we present a meta-study on the reproducibility and availability of research approaches for NLIDBs for real-world applications (see Section 3.2). While we were working on *DBPal*, Yu et al. [Yu+18b] published the *Spider* benchmark and challenge that fostered research on (cross-domain) NL2SQL and received a lot of submissions. Three years after the publication, we wanted to know whether the challenge really lead to (usable) progress, i.e., whether researchers and practitioners can use or build on the approaches submitted to the leaderboard of the shared task. Our analysis, however, lead to the conclusion that many of those approaches are not reproducible. We therefore propose changes for future benchmarks and tasks for (not only) NLIDBs and present a prototypical implementation called *UniverSQL* that makes these approaches easier to use in information access systems. We hope that this lowered barrier encourages (future) participants of these challenges to add support for actual usage of their submissions.

**Contributions:** We show that current benchmarks, especially the Spider challenge [Yu+18b] and the related challenges SparC [Yu+19b] and CoSQL [Yu+19a] are not sufficient to measure all relevant aspects and support the emergence of ready-to-use NLIDBs. Yet, to foster research not only on NLIDBs but on systems that integrate and use them, we publish an API called *UniverSQL* to integrate submissions to the challenges into research prototypes and existing systems. Its core functionality is a wrapper implementation to allow the execution of arbitrary queries on pre- or custom-trained models. We additionally provide two sample implementations of this wrapper for existing NL2SQL translators (*EditSQL* [Zha+19] and *IRNet* [Guo+19]). The code is published under an open source license. Finally, we provide an overview of the advantages and flaws of Spider and other benchmarks and provide ideas on how the evaluation of NLIDBs could advance.

As the final topic in this research field, we concentrated on natural language interfaces that allow not only information access but also manipulation. To guide users, and to make sure they only

perform intended actions, such interfaces are often implemented as so-called conversational agents (CAs) (i.e., chatbot-like interfaces to e.g., book a cinema ticket). However, developing such an agent requires both immense amounts of training data and NLP expertise. Therefore, we propose *CAT*, an approach that can be used to easily create CAs for transactional databases (see Section 3.3).

The key features of our approach are the use of weak supervision and data-awareness. We suggest a procedure to automatically generate training dialogues given a database and a set of transactions with only minimal manual overhead. We then use it to train a CA. As a major difference to existing CAs, agents synthesized by our approach are data-aware: We introduce a *data-driven dialogue policy* that leverages the data characteristics to request information from the user to minimize the number of dialogue turns, i.e., to fulfill a user request as quickly as possible. By incorporating the current data distributions of the database, we can reach markedly more efficient dialogues for many databases.

**Contributions:** We propose an approach called *CAT* that uses weak supervision to synthesize the required training data to train a state-of-the-art CA for a given OLTP database. Furthermore, our system provides an out-of-the-box integration of the resulting agent with the database. Agents created with our system are data-aware, i.e., decide which information should be requested from the user based on database distributions. We showcase our system by a demonstration scenario[2] with a fully synthesized CA for a movie database which allows a user to reserve tickets, cancel existing reservations and list movie theater screenings. We show both the creation of the agent using our system and the usage of the agent itself, and publish our approach as open source.

## 2.4. Personalized Summarizations of Text Collections

Our second research area deals with the use of personalized summaries for the exploration of unknown data collections. The idea is to offer the user a way to automatically produce summaries that cover certain aspects of a text collection. While these summaries can be used on their own, e.g., to add descriptions to a text collection, we argue that the possibility to interactively create multiple different summaries (e.g., with different degree of details or covered topics) can be very useful to understand large collections of topic-related documents and has real-world applications in journalism, medicine, and many more.

To make sure that the required level of customization is reached, we focus on providing the results at interactive speed. Key to our system is that the summarization model is refined by user feedback (i.e., marking important and irrelevant parts of a proposed summary) and called multiple times to improve the quality (i.e., the level of detail and focus relevant for the user) of the summaries iteratively. Such an approach was proposed by Avinesh P. V. S. and Meyer [AM17] where they use an integer linear program (ILP) to compute an optimal solution. Yet, since solving time grows exponentially with the number of constraints (e.g., the sentences to consider), each iteration may take hours. We therefore propose a sampling-based approach that builds on top of this, to guarantee interactive speeds even for large text collections to keep the user engaged in the process (see Section 4.1). As we show in our evaluation, our system can provide a similar quality level as the model that is working on the full corpus, but in a fraction of its runtime.

[2] See Section 10.5 and `https://link.tuda.systems/cat-video`

**Contributions:** We present *Sherlock*, an approach for interactive summarization of large text collections that uses sampling to provide interactive response times. As a main contribution, we propose a method to select the sample size based on iteration time thresholds and evaluate multiple different sampling strategies. Since choosing a suitable sample is not a trivial task, we study the effectiveness of multiple sampling strategies and the impact of the sample size on the summarization quality. To this end, we employ importance-based and stratified sampling, and benchmark them in a systematic experimental setup on different document collections.

As part of our research on multi-document summary (MDS), we noticed that there is a lack of diverse evaluation corpora for this task: large state-of-the-art corpora for training neural networks to create abstractive summaries are mostly limited to the news genre,[3] as it is expensive to acquire human-written summaries for other types of text at a large scale.

We therefore propose a novel automatic corpus construction approach (see Section 4.2) to tackle this issue, and present three new large open-licensed summarization corpora based on our approach that can be used for training abstractive summarization models. Our constructed corpora contain fictional narratives, descriptive texts, and summaries about movies, television, and book series from different domains. All sources use a creative commons (CC) license, hence we can provide the corpora for download. We provide our automatic construction approach as a ready-to-use framework to create custom corpora with desired parameters like the length of the target summary and the number of source documents that have to be summarized each. The main idea behind our automatic construction approach is to use existing large text collections (e.g., thematic wikis), automatically classify whether the texts can be used as (query-focused) multi-document summaries, and then align them with potential source texts. We show the usefulness of our automatic construction approach by running state-of-the-art summarizers on the corpora and through a manual evaluation with human annotators.

**Contributions:** We present a framework that can be used to create new summarization corpora and discuss reasonable choices for the parameters. We provide three new sample corpora created with our automatic construction pipeline (*FandomCorpora*). In a comprehensive evaluation based on these corpora (using traditional and neural network based methods) we validate that our pipeline is able to automatically create corpora of use for state-of-the-art summarizers. We make our code available as open source.

## 2.5. Information Extraction & Integration

As a third area, we provide ways to democratize information extraction and integration. This becomes relevant when data is scattered across different sources and there is no tabular representation that already contains all information needed. Therefore, it might be necessary to integrate different structured sources, or to even extract the required information pieces from text collections first and then to organize them. Only with these aggregated representations, one might really be able to create value and derive knowledge from the data. Yet, when manually done, these tasks require a lot of time and effort. In this part of the thesis, we thus explore how existing resources

[3] News corpora often completely rely on existing texts and are therefore relatively cheap to construct. Different approaches use, e.g., a teaser that is published with the article [PXS18] or even the headline as the summary [NGV12], and therefore require no manual annotation.

(like language models (LMs) and embeddings) can be leveraged to provide the required NLU for information extraction and integration tasks, in a way that they can be automized with a low overhead.

Since data is often stored in different sources, it needs to be integrated to gather a global view that is required in order to create value and derive knowledge from it. A critical step in data integration is therefore schema matching which aims to find semantic correspondences between elements of two schemata. When large amounts of data have to be integrated (e.g., when database systems of companies are unified after a merger, or to distill knowledge from large data lakes), tools to reduce the manual effort involved in schema matching are required. We propose a novel end-to-end approach for schema matching based on neural embeddings and compare it to existing solutions for the automatic determination of schema correspondences (see Section 5.1). The main idea is to use a two-step approach consisting of a table matching step followed by an attribute matching step. In both steps, we use embeddings on different levels, either representing the whole table or single attributes. Our results show that our approach is able to determine correspondences in a robust and reliable way and (compared to traditional schema matching approaches) can find non-trivial correspondences.

**Contributions:** We present and evaluate an end-to-end approach for schema matching using neural embeddings. As part of this, we propose and analyze different matchers on multiple levels (i.e., tables and columns) to identify a set of possible table and attribute correspondences. Furthermore, we compare them to existing approaches for automatic schema matching.

Based on these results, we then extended our scope: can we create a structured representation as requested by the user even when there are no existing tabular representations of the required contents, but just based on textual documents? Can we go beyond the fixed label sets of existing extraction approaches and fill arbitrary table columns that the user requested? Can a system do so without extensive computation and large overhead for adaption to support ad-hoc exploration? And will embeddings again be useful to find these non-trivial correspondences?

We hence propose a new system that allows users to interactively perform structured explorations of text collections in an ad-hoc manner (see Section 5.2). The main idea is to include user interaction to support extraction as requested and even ad-hoc SQL queries over text collections. We implement this using a new two-phased approach: first, a superset of information nuggets from the texts is extracted using existing extractors such as named entity recognizers. Then, the extractions are interactively matched to a structured table definition as requested by the user based on embeddings. In our evaluation, we show that this approach is thus able to extract structured data from a broad range of (real-world) text collections in high quality without the need to design extraction pipelines upfront. This approach makes it possible to automatically extract and organize relevant contents from large text collections, using a simple graphical interface. Users do not need programming skills for that. However, if they have, they can leverage their knowledge of SQL to pose more complex queries and compute aggregation results directly as part of this process.

**Contributions:** We propose *WannaDB*, a system that can execute SQL-like queries on text collections in an ad-hoc manner, both to extract facts from text documents, and apply filtering, aggregations and grouping. Our system can therefore directly produce tables stating information that is not explicitly mentioned in the documents, and hence not discoverable by pure extraction or search approaches. Our system implements a novel extraction and querying pipeline that

first extracts information nuggets independently of the query and then at runtime uses a novel interactive table filling approach that aims to map the information nuggets to a table specified by the user in the form of an attribute list or SQL query. For this table filling, it requests and generalizes feedback from the user. We conduct a wide range of experiments on text collections from different domains to validate our approach. Finally, we make our source code and the data sets used for evaluation available as open source.

## 2.6. Low Overhead Approaches for the Data Science Pipeline

As already outlined in Section 2.1, our approaches can be applied to perform typcial steps of the data science pipeline. A graphical representation how the steps interconnect can be found in Figure 2.2.

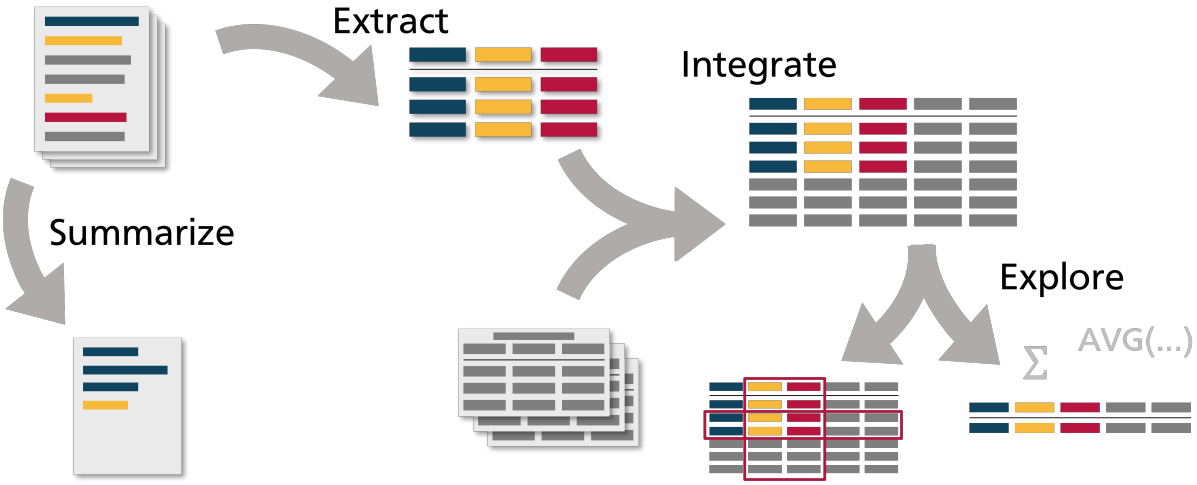



Figure 2.2.: Typical steps of the data science pipeline that our approaches can be applied to

Our methods can be used to learn about the contents of textual documents and whether they are suitable and relevant for an information need. Next, they can be applied to extract the relevant information into a structured format and integrate it with existing other data. Our approaches for data access can then be used to query and aggregate this structured data. Afterwards, a new information need might emerge where the tools can be applied again.

Our approaches have several advantages compared to existing ones: They can be used by non-experts and therefore alleviate the need for data scientists. This is also underlined by the focus on usability to allow a usage for even more people.

The approaches require less training or information than many other approaches and therefore enable cheap adaption and querying. This makes open exploration of data possible. Instead of focusing on single domain, we apply generalization as key principle, to allow for wide applications and therefore a democratization in many fields.

# 3. Natural Language Interfaces for Data Access & Manipulation

In a world where ever more data is generated, processed, and relied upon, it becomes continually more significant that data is not only accessible to a small group of people. Information can be contained in text, relational databases, knowledge graphs, and many other formats—but users do not want to deal with heterogeneous sources. They are interested in accessing information easily. The borders between structured and unstructured information keep blurring: when using Google for factual questions, infoboxes might show the answer without the need to open a search result. That result might even be wrapped in a generated sentence when voice search was used, and it is irrelevant whether the sentence was extracted from a web page or generated from a database.

On the other hand, there are good reasons why these different ways of storing information exist. Information access methods should leverage the capabilities of each approach, while providing convenient and ideally unified interfaces. With this goal in mind, natural language interfaces (NLIs) emerged as a data retrieval method, leveraging one of our most flexible and intuitive means of communication.

Relational databases are an essential type of information storage. They provide a structured representation that allows direct access to the relevant subset of information, as well as automatic computation of aggregated results that might be needed for a decision. To query them, users require knowledge of the domain, query language (e.g., SQL), and database schema. Contrarily, the vision for natural language interfaces for databases (NLIDBs) encompasses the ability of any user to interactively explore large datasets without help or extensive manual preparation work [JPP17]. As one of the biggest challenges, the application of NLIDBs requires the means to translate natural language (NL) into SQL queries (NL2SQL)—for a recent comprehensive overview of methods and open problems, refer to Kim et al. [Kim+20]. However, before such NLIDBs can be widely used as

one of many interfaces for information access (i.e., users can enter their information request using arbitrary words and get a correct answer without knowledge about the database), further research is needed.

In this thesis, we discuss two approaches that can be used by users without knowledge of SQL or the structure of a database to work with the information stored in it. First, we concentrate on SELECT-queries and propose an approach to build a read-only *NLIDB* called *DBPal* (see Section 3.1). Afterwards we propose an approach for building conversational agents that allow both accessing and manipulating structured data in a database called *CAT* (see Section 3.3). Both approaches can be applied to new domains with a low overhead, since they do not depend on large amounts of annotated training data. Instead, they apply a weak supervision approach, where existing data and templates are generalized and paraphrased by the computer to automatically generate sufficient amounts of training data in short time and at low costs. To complement this research, we present a meta-study on the reproducibility and availability of research approaches for NLIDBs for real-world applications in Section 3.2.

## 3.1. Natural Language Interfaces for Databases: DBPal

SQL, despite its expressiveness, may hinder users with little or no relational database knowledge from exploring and making use of the data stored in a database management system (DBMS). In order to effectively analyze such data, users are required to have prior knowledge of the syntax and semantics of SQL. These requirements set a high bar of entry for information access and data exploration, and have therefore triggered new efforts to develop alternative interfaces that allow non-technical users to interact with (their) data more conveniently. In addition to visual data exploration tools [e.g., Cro+15b; TAB], in particular NLIDBs seem to be such a highly promising alternative, since they enable users to pose questions in a concise and familiar manner.

Many approaches to construct these NLIDBs, however, require large amounts of domain-specific training data, and manual tuning. Furthermore, they focus solely on the translation and do not consider further measures to support the user when exploring the database through the interface. We therefore propose a complete system that enables users to build robust NL interfaces for different databases with low manual overhead.

**Publications:** The work on *DBPal* was mainly published in two peer-reviewed publications. The general idea and a prototypic implementation was published in the demo paper "Fuat Basik, Benjamin Hättasch, Amir Ilkhechi, Arif Usta, Shekar Ramaswamy, Prasetya Utama, Nathaniel Weir, Carsten Binnig, and Ugur Çetintemel. 'DBPal: A Learned NL-Interface for Databases'. In: *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*. ACM, 2018" (see Chapter 7). It contained a system architecture for both the translation component and the training data generation. We concentrated on end-users without knowledge of the database structure and discussed which features a graphical user interface suitable for them should have. We implemented such an interface prototypically and explained both the approach and its exemplary usage in the paper and a video.
The full approach, including improved components and an extensive evaluation, was then published as "Nathaniel Weir, Prasetya Utama, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Nadja Geisler, Benjamin Hättasch, Steffen Eger, et al. 'DBPal: A

Fully Pluggable NL2SQL Training Pipeline'. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. ACM, 2020" (see Chapter 8). In the paper, we analyze why available training data for NL2SQL may not be sufficient and why automatic approaches are needed to create data for adaption of translators. We therefore propose a system that augments existing deep learning techniques in order to improve the performance of models for NL2SQL translation. More specifically, we present a novel training pipeline that automatically generates synthetic training data in order to improve overall translation accuracy, increase robustness to linguistic variation, and specialize the model for the target database. We show that our training pipeline is able to improve both the accuracy and linguistic robustness of state-of-the-art NL2SQL translation models. Moreover, we present a new benchmark to explicitly evaluate the linguistic robustness. In addition to these two publications, we published preprints of our research on ArXiv [Uta+18; Wei+19b].

**Contributions of the author:** I was a co-author of the publications. My central responsibilities in this project comprised literature research and description, making improvements to the training data generation component, conducting experiments and confirming results of previous ones, integrating *DBPal* into a speech assistant, and publishing the *Paraphrase Benchmark* corpus that was created as part of the project. I wrote parts of the manuscripts of the publications and helped to refine the remainder. All authors agree with the use of the publication for this dissertation.

### 3.1.1. Our Approach

In the last years, the number of deep learning approaches for translating natural language to SQL grew [KK23; Kim+20; Qin+22]. The usage of foundational large language models (LLMs) for that task gained traction, yet, zero-shot approaches (e.g., prompting *ChatGPT*) do currently not provide the same quality as existing trained models, as benchmarks like Bird[1] show. Therefore, accompanying learned models are needed to pre- and post-process inputs and outputs or to generate advanced prompts adapted to the inputs. These require suitable training and evaluation data. Furthermore, invoking closed models for translation may not always be possible, be it because of the associated costs or privacy issues.

More traditional approaches for NL2SQL translation [Iye+17; Wan+15; XLS17] can be used locally and at lower costs, but they rely on supervised learning approaches that require substantial amounts of training data, too. The same holds for open source LLMs, which currently provide an inferior translation quality compared to the widely used closed models [Sun+23], but can be drastically improved with fine-tuning [Gao+23].

This training data is costly to acquire, in particular since many approaches require domain-specific data. As such, additional manual effort is needed for each new database schema, which severely limits the portability of these approaches to new domains. In order to address this fundamental limitation, we have built DBPal as a complete system that enables users to build robust NL interfaces for different databases with low manual overhead.

At its core, DBPal implements a novel training pipeline for NLIDBs that synthesizes its training data using the principle of *weak supervision* [Cra+00; Deh+17]. The basic idea of weak supervision is

[1] `https://bird-bench.github.io/`

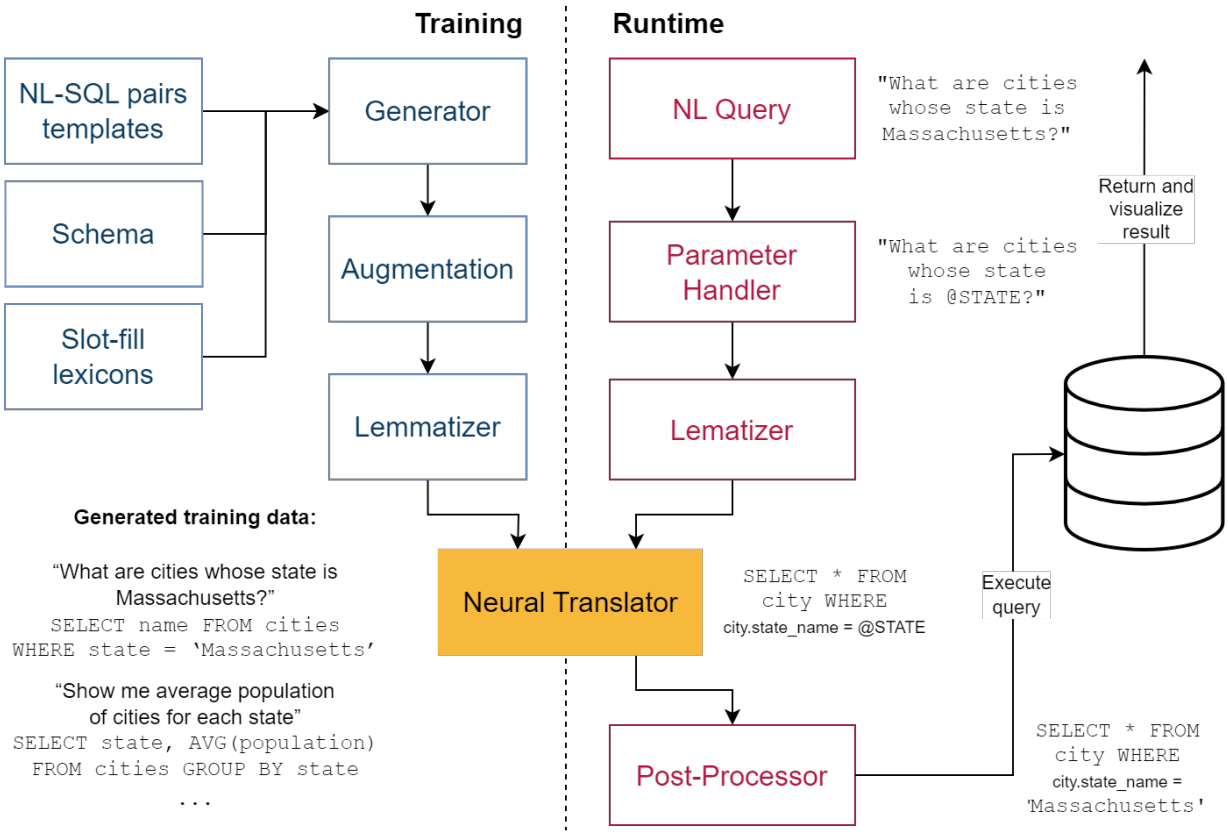



Figure 3.1.: DBPal training pipeline, and lifecycle of a query at runtime

to leverage various heuristics and existing datasets to automatically generate large (and potentially noisy) training datasets instead of handcrafting them.

In its basic form, only the database schema is required as input in order to generate a large collection of pairs of natural language queries and their corresponding SQL statements, that can then be used to train our language translation model. In order to maximize our coverage across natural language variations, we use additional input sources to automatically augment the training data using a collection of techniques. One such augmentation step is an automatic paraphrasing process using an off-the-shelf paraphrasing database [PC16].

Figure 3.1 shows how our fully functional prototype is both trained and used at runtime. At the core of our prototype is a *neural translator*, trained by DBPal's pipeline, which translates incoming NL queries into SQL queries. Importantly, our fully pluggable training pipeline is agnostic to the actual translation model; that means DBPal is designed to improve the accuracy of existing NL2SQL deep learning models (e.g., SyntaxSQLNet [Yu+18a]) by providing specific training data for a given database schema.

**Training Phase:** During the training (or fine-tuning) phase, our training pipeline provides existing NL2SQL deep learning models with large corpora of synthesized training data. This training pipeline, described further the next section, consists of three steps to synthesize the training data: (1) generator, (2) augmentation, and (3) lemmatizer. Once training data is synthesized by DBPal's

pipeline, it can then be used (potentially together with existing manually curated training data) to train existing neural translation models that can be plugged into the training pipeline.

**Runtime Phase:** The runtime phase can leverage this trained model, as shown on the right-hand side of Figure 3.1. The *Parameter Handler* is responsible for replacing the constants in the input NL query with placeholders to make the translation model independent of the actual database and help to avoid retraining the model if the underlying database is updated. For example, for the input query shown (i.e., "What are cities whose state is Massachusetts?"), the *parameter handler* replaces "Massachusetts" with the appropriate schema element using the placeholder `@STATE`. The *lemmatizer* then combines different variants of the same word to a single root. For example, the words "is", "are", and "am" are all mapped to the root word "be". Then, the *neural translator* works on these pre-processed and generalized NL input queries and creates output SQL queries, which also contain placeholders. In the example, the output of the Neural Translator is: `SELECT name FROM cities WHERE state = @STATE`. Finally, the *post-processor* replaces the placeholders with the actual constants such that the SQL query can be executed. The result is then presented to the user using a tabular visualization. We showcase the exemplary usage of the system in Section 7.4.

Additionally, the pre-processor components and the post-processor are also involved in order to handle complex SQL queries such as JOINS and nested statements. The details of the pre- and post-processing are explained further in Sections 8.4 and 8.5.

**Interactive Query Auto-Completion:** In addition, we propose a component for real-time auto-completion and query suggestion to help users who may be unfamiliar with the database schema or the supported query features. This may also increase translation accuracy by leading them to enter less ambiguous queries. Consider a user exploring a geographical information database and starting to type "show me the names "—at this point, the system suggests possible completions such as "of states", "of rivers", or "of cities" to make users aware of the different options they have. The core of the proposed auto-completion feature is a language model based on the same sequence-to-sequence model and trained on the same dataset as the query translator. More details can be found in Section 7.3.3.

### 3.1.2. Training Pipeline

In the following, we describe our training pipeline and focus in particular on the data generation framework. The details of the full training pipeline are explained further in Section 8.3.

**Generator:** In the first step, the *generator* uses the database schema along with a set of seed templates that describe typical NL-SQL pairs to generate an initial training set. In the second step, *augmentation*, the training data generation pipeline then automatically adds to the initial training set of NL-SQL pairs by leveraging existing general-purpose data sources and models to linguistically modify the NL part of each pair.

The main idea is that each seed template covers a typical class of SQL queries (e.g., a `SELECT-FROM-WHERE` query with a simple predicate). Composing the seed templates is only a minimal,

one-time overhead, and all templates are independent of the target database (i.e., they can be reused for other schemas). Furthermore, in DBPal, we assume that the database schema provides human-understandable table and attribute names, but the user can optionally annotate the schema to provide more readable names if required; deriving readable schema names automatically is an orthogonal issue.

The schema information is then used to instantiate these templates using table and attribute names. Additionally, manually predefined dictionaries (e.g., to cover synonyms) can be used to instantiate simple variations of NL words and phrases (e.g., "Show me" and "What is" for the `SELECT` clause). We propose a list of approximately 100 seed templates. A typical training set that can be generated from these templates contains around one million NL-SQL pairs for a simple, single-table database schema and around two to three million for more complicated schemas.

**Augmentation:** A core aspect of our pipeline is the *augmentation* step, that automatically expands the training data produced by our generator in order to offer more accurate and linguistically robust translations. During augmentation, the training data generation pipeline automatically adds new NL-SQL pairs by leveraging existing general-purpose data sources and models to linguistically vary the NL part of each pair. The goal of the augmentation phase is thus to cover a wide spectrum of linguistic variations for the same SQL query, which represent different versions of how users might phrase the query in NL. This augmentation is the key to make the translation model robust and allows DBPal to provide better query understanding capabilities than existing standalone approaches. Section 8.3.2 describes this process in more detail.

**Lemmatization:** Finally, in the last step of the data generation procedure, the resulting NL-SQL pairs are lemmatized to normalize the representation of individual words. During this process, different forms of the same word are mapped to the word's root in order to simplify the analysis (e.g., "cars" and "car's" are replaced with *car*). The same lemmatization is applied at runtime, too.

### 3.1.3. Evaluation: Paraphrase Bench

To measure how well a model can cope with linguistic variations, we crafted an open-source benchmark called *ParaphraseBench*.[2]

The schema of our new benchmark models a medical database comprised of hospital patients with attributes such as name, age, and disease. In total, the benchmark consists of 399 carefully crafted pairs of NL-SQL queries.

To test the linguistic robustness of the given translation model, queries are grouped into one of the following categories depending on the linguistic variation that is used in the NL query: *naïve*, *syntactic paraphrases*, *morphological paraphrases*, *semantic paraphrases*, and *lexical paraphrases*, as well as a category where queries have some *missing information*. These categories are formulated along the guidelines of paraphrase typologies discussed in Vila, Martí, and Rodríguez [VMR11] and Bhagat and Hovy [BH13]. While the NL queries in the *naïve* category represent a direct translation of their SQL counterpart, the other categories are more challenging: *syntactic paraphrases* emphasize structural variances, *lexical paraphrases* pose challenges such as synonymous words and phrases,

[2] `https://link.tuda.systems/paraphrase-bench`

*semantic paraphrases* use changes in lexicalization patterns that maintain the same semantic meaning, *morphological paraphrases* add affixes, apply stemming, etc., and the *missing* category includes implicit references to concepts. Exemplary queries for the categories are:

**Naïve** "What is the average length of stay of patients where age is 80?"

**Syntactic** "Where age is 80, what is the average length of stay of patients?"

**Morphological** "What is the averaged length of stay of patients where age equaled 80?"

**Lexical** "What is the mean length of stay of patients where age is 80 years?"

**Semantic** "On average, how long do patients with an age of 80 stay?"

**Missing Information** "What is the average stay of patients who are 80?"

Unlike other benchmarks that test for exact syntactic match of SQL queries, our benchmark tests instead for semantic equivalence. Since the test set is (relatively) small (i.e., 57 queries per category), we manually enumerated possible semantically equivalent SQL query answers. However, if the benchmark were to be extended, one could use an equivalence checker (e.g., *Cosette* [Chu+17]) to verify correctness. We think that our benchmark is a good complement to existing, larger benchmarks (see Section 3.2).

### 3.1.4. Key Findings

We will now show that the presented training pipeline is able to improve the performance of existing NL2SQL translation techniques. A complete evaluation including all details and additional microbenchmarks can be found in Section 8.6.

We first compare our proposed augmentation techniques to the training process using *SyntaxSQLNet* [Yu+18a] with the well-known *Spider* [Yu+18b] benchmark that consists of over $10,000$ NL questions paired with the corresponding SQL queries. The benchmark contains $200$ database schemas, each of which has several tables, representing real-world database deployments. The data in the benchmark is very diverse and spans $138$ distinct domains (e.g., automotive, social networking, geography). In addition to the diverse data, the corresponding SQL queries contain almost all common SQL patterns, including nested queries. Based on the complexity of the corresponding SQL query (i.e., the number of SQL components), each question is assigned a difficulty level. Further information on the dataset can be found in Section 3.2.1. *SyntaxSQLNet* is a state-of-the-art deep learning model that uses pre-trained *GloVe* word embeddings [PSM14] when parsing the words in the input sentences. Using *GloVe* embeddings already allows the model to handle variations of individual words efficiently.

In this benchmark, accuracy is measured by computing the number of correctly translated NL phrases divided by the total number of queries. A query is deemed to be correctly translated only if it exactly matches the provided "gold standard" SQL query for the NL input, without allowing for semantically equivalent answers. Unlike other datasets, *Spider* uses different databases (i.e., schemas and data) for training and testing (i.e., a database schema is used exclusively for either training or testing, but not both). This allows us to evaluate how well the model will generalize to new domains.

| | Easy | Medium | Hard | Very Hard | Overall |
|---|---|---|---|---|---|
| SyntaxSQLNet | 0.445 | 0.227 | 0.231 | 0.051 | **0.248** |
| DBPal (Train) | 0.472 | 0.300 | 0.252 | 0.107 | **0.299** |
| DBPal (Full) | 0.480 | 0.323 | 0.279 | 0.122 | **0.317** |

(a) *Spider* benchmark

| | Naive | Syntactic | Lexical | Morph. | Semantic | Missing | Mixed | Overall |
|---|---|---|---|---|---|---|---|---|
| SyntaxSQLNet | 0.281 | 0.228 | 0.070 | 0.175 | 0.175 | 0.088 | 0.140 | **0.165** |
| DBPal (Train) | 0.930 | 0.333 | 0.404 | 0.667 | 0.228 | 0.088 | 0.193 | **0.409** |
| DBPal (Full) | 0.947 | 0.632 | 0.544 | 0.667 | 0.491 | 0.158 | 0.298 | **0.531** |

(b) *ParaphraseBench* benchmark

Table 3.1.: Accuracy on the *Spider* and *Paraphrase* benchmarks of a *SyntaxSQLNet* model trained with our approach compared to the original model without fine-tuning by *DBPal*. Values between 0 and 1, higher is better. Our approach can outperform the original model for all difficulty levels and linguistic variation categories.

Table 3.1a shows the accuracy for *SyntaxSQLNet* using the Spider dataset for three different configurations. First, as a baseline, we show the performance of the base *SyntaxSQLNet* model trained using the data from Spider's training set. The DBPal (Train) configuration uses the *SyntaxSQLNet* model finetuned with our training pipeline, i.e., with additional synthetic data generated by DBPal using the schemas of the training set in Spider only. Finally, the DBPal (Full) version uses the schemas from both the training and test set of Spider to generate additional synthetic training data. Note, however, that DBPal never sees actual NL-SQL pairs from the test set during the training process, only the schemas (in the DBPal (Full) configuration).

As shown, both configurations of DBPal improve upon the original model across all difficulty levels. In the DBPal (Train) case, we see that, with the addition of synthetic training data generated only using schema information from the training set, DBPal is already able to outperform the baseline model. This is due to the fact that our novel training pipeline is able to supplement the existing training data with additional query patterns (e.g., nested subqueries) that are not present (or numerous enough) in the training data. As shown, this helps substantially for the harder queries, with DBPal being able to outperform the baseline by more than $2\times$ for the "very hard" category due to the fact that the training pipeline introduces new query patterns (e.g., nested queries) to the model.

In general, DBPal (Full) is able to leverage additional query patterns from the synthetic data generation pipeline that are specific for the test schemas. With this information, DBPal (Full) is able to generate training examples that provide the model with additional information (e.g., table names, column names, column values) that is specific to test databases. As shown in Table 8.2, the added synthetic data for the test schemas in Spider when using DBPal (Full) is able to offer additional performance improvement over DBPal (Train). More concretely, with the help of the additional generated training data, we can further improve translation accuracy across all query difficulties of Spider by $15 - 27\%$. This shows that even though generalizing models are very useful to support different databases without manual training, an automatic adaption approach like the one proposed by us—that adds information on the target scheme during training—can further improve the translation quality.

To test the robustness of our system against linguistic variations, we test it on the *ParaphraseBench* introduced in Section 3.1.3. The results can be found in Table 3.1b. Again, the model trained with our pipeline can outperform the baseline model on all categories (by about $0.25$). Additionally incorporating schema information of the target databases can increase the translation accuracy even further, from $0.165$ to $0.531$.

In general, the results of our training data augmentation fall into two categories. On one hand, there are query pattern categories (e.g., Naive, Morphological) where the improvement can be mainly attributed to the additional patterns introduced with our training data generation baseline. Here, additional target schema knowledge provides little to no benefit.

For other categories (e.g., semantic, missing), target schema knowledge makes a large difference, and may lead to a doubled accuracy. Here, we find that schema information is particularly helpful because it allows the model to learn complex, domain-specific NL mappings. For example, consider the example semantic query: "On average, how long do patients older than 80 stay?" Clearly, the semantic meaning of the phrase "older than" refers implicitly to the "age" attribute of the patient, but this would not be easy to derive from a generic training set. However, by providing training data specifically generated from the target schema, DBPal is able to help the model to better learn these mappings.

In summary, our evaluation shows that our approach is suitable to boost the quality of existing NL2SQL approaches. Furthermore, we can show that the automatic generation of new NL-SQL-pairs for a specific schema provides an improvement over automatically generating generic training data.

## 3.2. Natural Language Interfaces for Databases: Meta Study on other Approaches

While we were working on *DBPal*, Yu et al. [Yu+18b] published the *Spider* benchmark and challenge,[3] that fostered research on (cross-domain) NL2SQL. The challenge received over $100$ submissions in different categories, quickly raising the accuracy for translation without value prediction from about $25\,\%$ to over $70\,\%$ on the test set of the challenge in the following years. Two similar challenges for context-depending/multi-turn translation (SParC) [Yu+19b] and conversational text-to-SQL translation (CoSQL) [Yu+19a] followed, receiving not as many submissions as Spider, but still showing a substantial improvement over the baselines published with the datasets. But how useful are those submissions when one now wants to build a system that requires NL2SQL? That was something we wanted to find out in a meta study:

**Publication:** We published this work as "Benjamin Hättasch, Nadja Geisler, and Carsten Binnig. 'Netted?! How to Improve the Usefulness of Spider & Co'. In: *Proceedings of the Second International Conference on Design of Experimental Search & Information REtrieval Systems, Padova, Italy, September 15-18, 2021*. Volume 2950. CEUR Workshop Proceedings. CEUR-WS.org, 2021" (see Chapter 9). In the fullpaper, we analyzed the usefulness of submissions to the *Spider*, *SParC*, and *CoSQL* challenges for future research and found that these are not sufficient to measure all relevant aspects and support the emergence of ready-to-use NLIDBs. Additionally, we presented *UniverSQL*,

[3] Yale Semantic Parsing and Text-to-SQL Challenge: `https://yale-lily.github.io/spider`

a prototypical implementation of an interface between submission entries and custom systems requiring NL2SQL. As a final contribution, we provided ideas how benchmarks and shared tasks for NLIDBs should be adapted in the future to focus not only on high scores but also support real usage of submissions.

**Contributions of the author:** I was the lead author of the publication and thus responsible for structuring both the study and the further parts of the publication, conducting the analysis and writing the manuscript. The co-author Nadja Geisler supported in confirming the experiments and provided parts of the ideas on how to improve challenges in the future. The prototypical implementation of the interface was partly done by students under the supervision of Nadja Geisler and me. The co-author Carsten Binnig contributed invaluable feedback. All authors agree with the use of the publication for this dissertation.

### 3.2.1. Benchmarks for NLIDBs

Modern data driven approaches would not be possible without big amounts of data, but curating and annotating it is out of scope for many researchers. Thus, public datasets and benchmarks play an important role for the development and evaluation of NL2SQL translators. We will therefore give a quick overview of them here:

The Spider challenge [Yu+18b] has become one of the standard evaluations for NLIDBs since its publication in 2018, being cited over $620$ times by the publication time of this thesis. The main leaderboard of the shared task has more than $80$ entries (as of October 2023). The dataset aims to surpass most existing datasets in size by at least one order of magnitude. At the same time, it covers a diverse set of simple and complex SQL queries. This provides the necessary basis for data-driven systems to translate joins, nestings etc., and challenges them to do so to achieve good performance on the development and test data splits. One has to highlight the manually annotated and high-quality data, which deservedly makes it the currently most important benchmark for NL2SQL translators.

Alongside the dataset, Spider provides a shared task that encourages building NLIDB systems capable of generalizing to new databases and performing well across domains, without the need to expensively create a new training dataset for each database. The split ensures that each database occurs in exactly one set (training, development, and test). This provides a concrete task description and evaluation process, allowing accurate and comparable measurements of success. The Spider shared task encourages the submission of models to show up in the leaderboard. There are two variants: the original task does not check value accuracy, but there is also a leaderboard for systems that handle/predict values (not just queries with placeholders).

*SparC* [Yu+19b] is the multi-turn variant of *Spider*. It deals with cross-domain semantic parsing in context and is comparable to *Spider* in size, complexity and databases. However, queries are arranged in user interactions, providing dialogue-like context. Therefore, it is not sufficient to just translate the current NL utterance into SQL, but information from previous queries has to be taken into account. Analogous to *Spider*, *SparC* features a leaderboard for variants with and without value handling. Although this challenge was published just nine months after the Spider challenge, it received considerably fewer submissions so far ($22$ on the main leaderboard as of October 2023).

| | Spider (-v) | Spider (+v) | SparC | CoSQL |
|---|---|---|---|---|
| **Entries** | 62 | 7 | 17 | 10 |
| **Diff. appr.** | 51 | 5 | 15 | 8 |
| **- Publications** | 36 (58 %) | 6 (86 %) | 8 (47 %) | 9 (90 %) |
| **- Code** | 20 (32 %) | 4 (57 %) | 4 (24 %) | 5 (50 %) |

Table 3.2.: Analysis of the leaderboard entries for Spider (with (+v) and without (-v) value prediction), SparC & CoSQL. We checked how many different approaches are presented, how many of them reference a publication and how often there is code to at least try to reproduce the approach. Originally printed in Hättasch, Geisler, and Binnig [HGB21].

| | Spider (-v) | Spider (+v) | SparC | CoSQL |
|---|---|---|---|---|
| **Repositories** | 15 | 2 | 4 | 5 |
| **- Empty?** | 2 (13 %) | 0 (0 %) | 0 (0 %) | 1 (20 %) |
| **- Code?** | 13 (87 %) | 2 (100 %) | 3 (75 %) | 4 (80 %) |
| **- Checkpoints** | 9 (60 %) | 2 (100 %) | 3 (75 %) | 2 (40 %) |
| **- Own data?** | 2 (13 %) | 0 (0 %) | 0 (0 %) | 0 (0 %) |

Table 3.3.: Analysis of the available repositories for the different challenges. We report whether the repositories are empty or contain code, whether checkpoints/pre-trained models are provided for download and whether the usage of this approach on own data/tables is in some way prepared. Originally printed in Hättasch, Geisler, and Binnig [HGB21].

*CoSQL* [Yu+19a] takes the challenge to the level of a real conversational agent, but one, that should get its information from a database and not resort to world knowledge incorporated in a (large) language model. It consists of both dialogues and annotated SQL queries simulating real-world DB exploration scenarios. Therefore, the system has to maintain a state. *CoSQL* defines several challenges, the simplest one mainly adds further context to interpret compared to *SparC*, the other ones cover generation of suitable responses and intention detection/classification. At the time of our analysis, the challenge was public for around 20 months.

Besides these challenges, there is another benchmark that is commonly used: The *WikiSQL* Benchmark [ZXS17] is a large dataset (though smaller than Spider) that also features a leaderboard. Unfortunately, it consists only of a small number of unique query patterns [Fin+18] (in fact, half of the questions in the dataset are generated from one single pattern). In particular, it contains neither joins nor nestings. Furthermore, the NL questions are often low quality (i.e., many are grammatically incorrect), some do not have a proper semantic meaning and make little sense when read by humans, and some NL questions do not have the same meaning as the associated SQL query.

### 3.2.2. Key Findings

As mentioned above, the *Yale* challenges were widely adopted for evaluating NLIDBs. But how well do the growing scores on their leaderboards really reflect the progress in the community for this task? In our study in June 2021, we evaluated the state of the submissions, particularly with regard to how reproducible the submissions are and whether they can be used for real world translation tasks.

We print two tabular representations of our analysis (as of June 2021) here in Tables 3.2 and 3.3. We report results on both variants of the *Spider* challenge, as well as the main task of the other two challenges, since the leaderboards of the other variants contained at maximum two entries (all without code) except for the baselines provided by the authors of the challenges. In the following, we will outline our key results/interpretations of the state, for the details we refer to Chapter 9.3.

Taking the different challenges and variations together, there were 96 leaderboard entries in June 2021. Some of them are only small variations of the same system, nevertheless, this boils down to 79 different approaches. 59 or 61.5 % of the entries of those approaches were published in some way, the remaining approaches were anonymous or contained only names of authors or institutions (so far). For 40 (42 %) of the submissions, a link to code was provided, yet, some repositories are empty, or the link was invalid.

In total, we analyzed 26 repositories. Three of them (12 %) were completely empty, the others except for one contained code. More than half of them (16 or 62 %) provided pre-trained models for download. However, for most of the approaches, there is no preparation for the use of the models outside the evaluation scripts at all.

To summarize, only about a third (33) of the leaderboard entries had at least some code that could be used as a starting point for reproduction. Even worse, this was not evenly spaced, e.g., only for two of the top ten submissions to the main *Spider* challenge (and for four of the top twenty) code was provided. Only two submissions [LSX20; Shi+20] to the original *Spider* task provide a Juypter Notebook or a command line interface such that users can translate own queries.

Overall, we therefore had to conclude that reproducibility of the approaches submitted to the leaderboards of all challenges is at best mediocre, which is in line with problems of the community and especially research in computer science where reproducibility is still a challenge and publishing code and artifacts that allow others to redo the experiments is still optional.[4]

### 3.2.3. Improvements

The results show that there is still room for improvement, having leaderboards with high scores on it does not really "solve" the task. But how could this be improved?

The fast-moving (research) world makes it hard to really invest time to "finish" research approaches and bring them to a state where they can be easily reproduced and used by others. Yet, doing this can boost both the usage of the proposed approaches for real-world applications, and increase the productivity of the overall community that has to spend less time trying to figure how to run other approaches for comparison or to build upon them.

In our perspective, this issue can (and should) be tackled two-fold: On one hand, authors of a shared task can make it easier for participants to turn their approaches and models into directly usable systems, by already providing a suitable framework. On the other hand, participants should be encouraged to really make that effort. This could be done by badges or special leaderboards – but ultimately also by requiring publishing information for reproduction as part of the submission to a shared task.

[4] Despite efforts like reproducibility badges in the ACM Digital Library: `https://www.acm.org/publications/policies/artifact-review-and-badging-current`

**Making it easy to turn approaches into systems:** To foster research not only on NLIDBs but on systems that integrate and use them, we published an API called UniverSQL[5] to integrate submissions to the challenges into research prototypes and existing systems. Its core functionality is a wrapper implementation to allow the execution of arbitrary queries on pre- or custom-trained models, more details can be found in Section 9.4). We additionally provide two sample implementations of this wrapper for existing NL2SQL translators (EditSQL [Zha+19] and IRNet [Guo+19]). The code is published under an open source license.

**Encouraging or enforcing reproducibility:** First and foremost, this requires publication of code. It is fine that submissions are anonymous until the approach was reviewed and published. But we advocate that once names are revealed, it should also be necessary to reference publication and code. Authors of a challenge set the requirements for submissions to be included in a leaderboard—and they should take advantage of that. To make that code then really usable, often already small actions would help: pinning versions of dependencies (especially machine learning libraries often introduce breaking changes in just months), run the code on a second machine under a different username, add an installation script to download required external data or add environment variables for configuration. Each of these steps can make it substantially easier to run foreign code (or your own after a while). We therefore argue that shared tasks like Spider should require this in the future for submissions to their leaderboards, and find it a great pity that most of the current submissions are difficult to reproduce and even more difficult to utilize for further research.

Finally, we argue that future benchmarks should go beyond that and take the user's perspective into account. One way to do so could be end-to-end benchmarks that do not only evaluate the translation accuracy but the real performance in a data exploration task from input to the output (*SparC* and especially *CoSQL* do this to some extent). But there are many other interesting questions: We can measure the accuracy of a system like an NLIDB, but what accuracy should we strive for? Are all errors equally bad? Can a slightly wrong translation still be sufficient? What is the influence of a suboptimal translation? Will the user be satisfied by a system with $100\,\%$ translation accuracy? Or do they expect something that cannot be accomplished even by perfectly working systems? Answering such questions is hard, it can probably not always be automated, and it is difficult to frame the answer as a bunch of numbers. Yet, a framework to assess a system with respect to these kinds of questions would help to better decide on which improvements it is worth to focus. We therefore hope that this user perspective will be considered more regularly in computer science research—not as a separate field of research, but an integral part to drive research in a direction that is suitable to support humans best in whatever they want to accomplish.

## 3.3. Conversational Agents: CAT

Until now, we only considered read access to a database and thus NL2SQL translation of `SELECT` queries. As the final part of this research direction, we will now propose a low overhead approach for a system for editing/manipulating data stored in a database using natural language.

Online transaction processing (OLTP) databases are often the backbone for applications such as hotel room or cinema ticket booking applications. These applications already allow manipulating

[5] `https://link.tuda.systems/univerSQL`

the data in the database in a safe and designed way—but require users to remember application-specific commands or the correct usage of user interfaces. Natural language interfaces can provide an intuitive alternative to interact with such applications by allowing users to directly express their needs. Moreover, consumer products like Amazon Alexa or Apple Siri further raise the expectations of customers to interact using natural language. As a result, companies began developing conversational agents for supporting simple tasks or even basic business processes. For instance, a customer of an insurance company could report a claim or check the status of an existing report using such a CA.

However, developing a conversational agent (i.e., a chatbot-like interface) to allow end-users to interact with an application using natural language requires both immense amounts of training data and natural language processing (NLP) expertise. This motivates our approach, which can be used to easily create conversational agents for transactional databases.

**Publication:** We published this work as "Marius Gassen, Benjamin Hättasch, Benjamin Hilprecht, Nadja Geisler, Alexander Fraser, and Carsten Binnig. 'Demonstrating CAT: Synthesizing Data-Aware Conversational Agents for Transactional Databases'. In: *Proc. VLDB Endow.* 15.12 (2022)" (see Chapter 10). In that demo paper, we show how for a given OLTP database, one can use weak supervision to synthesize the required training data to train a state-of-the-art conversational agent, allowing users to interact with the OLTP database. The approach and the web app we provide to bundle it, allow creating such a CA both without immense amounts of training data and NLP expertise. As a major difference to existing conversational agents, agents synthesized by our approach *CAT* are data-aware. This means that the agent decides which information should be requested from the user based on the current data distributions in the database, which typically results in markedly more efficient dialogues compared with non-data-aware agents.

**Contributions of the author:** The first ideas and basic implementation of this approach were developed in a master thesis by Marius Gassen, co-supervised by Benjamin Hilprecht, Carsten Binnig and me. Together with Marius Gassen and Benjamin Hilprecht, I was then co-author of the first (unsuccessful) publication attempt to VLDB 2020. Afterwards, I took the lead of revising the paper and was thus responsible for completely reworking the manuscript, creating new graphics and a new demo video,[6] and coordinating the revision and submission process. The co-authors Nadja Geisler, Alexander Fraser and Carsten Binnig contributed invaluable feedback. All authors agree with the use of the publication for this dissertation.

### 3.3.1. Problem

Developing a task-oriented dialogue system for a given OLTP application (e.g., allowing users to buy a movie ticket) is a daunting task because this not only requires large amounts of annotated training data (i.e., actual dialogues between users and the system) for every application but also a manual integration with the existing database.

For instance, creating a conversational agent for a cinema ticketing system requires training data consisting of *user utterances* (e.g., "I want to reserve four seats tonight"), along with filled slots (e.g.,

[6] `https://link.tuda.systems/CAT-video`

`no_seats`=4) and annotated *user intents* (e.g., "reserve seats" or "inform about available shows"). These dialogues, however, are expensive to gather and annotating them is a large manual error-prone effort which requires extensive domain-knowledge. Worse, neither the training dialogues nor the integration with the existing database can be reused for a different domain.

Another drawback of existing approaches to build task-oriented dialogue systems is the lack of integration between the task-oriented dialogue system and the OLTP database, which is often the backbone of the business process. In current systems, a large amount of information must be provided manually even though it is already implicitly available in the database (for instance the required slots/attributes, the associated data types, the affected tables, etc.). Moreover, existing dialogue systems learn the order and types of information to request from the user purely from the manually created user dialogues. Not taking the data characteristics into account results in inefficient dialogues, as we describe below.

### 3.3.2. Idea

The main idea is to use weak supervision to synthesize the required training data to train a state-of-the-art conversational agent from a given OLTP database and a set of transactions (i.e., an OLTP workload with user-defined functions) with only minimal manual overhead. We implement this approach in a system we call *CAT*. Given a database and a set of transactions, the user only has to provide a few example formulations for each intent instead of numerous annotated example dialogues. Using a data-driven simulation, our approach generates annotated dialogues of possible user interactions from those intents, which can then be leveraged to train a conversational agent. This alleviates the extensive process of manually creating dialogues, which has to be repeated for every domain and database.

To perform correct transactions based upon the interaction with the user, the conversational agent needs to fill in correct values. To do this, it is often required to uniquely identify entities of the database. For instance, in order to book cinema tickets, the corresponding *customer ID* is required. Often the customer will not have the unique ID at hand but only information such as their name or address. We propose a data-aware approach; i.e., one that considers the data characteristics at runtime to (1) deal with incomplete information (e.g., a customer who cannot remember an ID) and (2) request the most suitable information to narrow down the set of candidates as quickly as possible. Different from existing conversational approaches which take a pure learning-based approach to determine what to ask for, *CAT* uses information such as database statistics (e.g., selectivities). For example, once the user provided their name, the agent might ask them for the city they live in, knowing that based on the entries in the database this is sufficient to uniquely identify the *customer ID* (while another name requires a different attribute to narrow down the options).

### 3.3.3. Architecture

The goal of our system is to synthesize conversational natural language interfaces for database transactions while avoiding the shortcomings of existing task-oriented dialogue systems. To address these problems, our approach leverages the information about a given database and a set of transactions: this is done for training data generation with weak supervision, but also at runtime

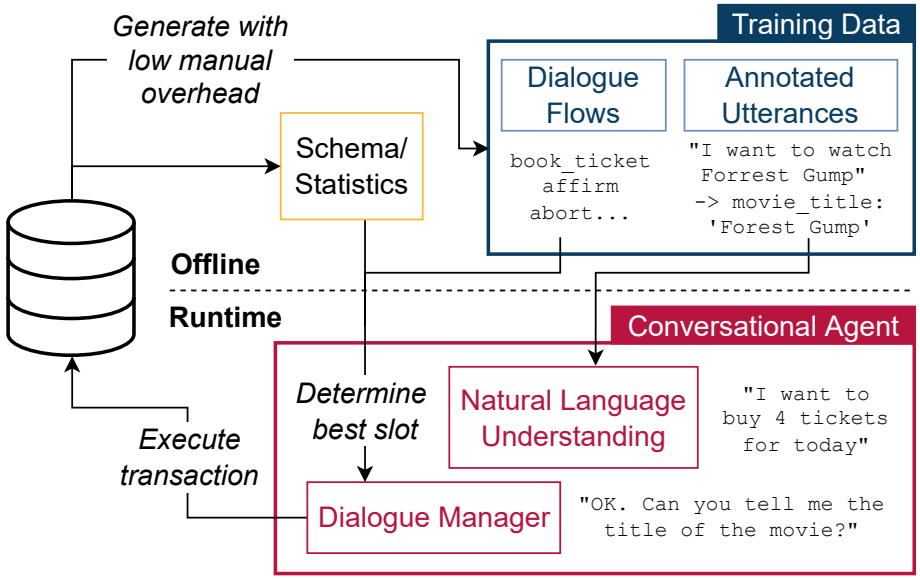



Figure 3.2.: Overview of our conversational agent creation framework *CAT*, showing both the creation and the usage of an agent. Published in Gassen et al. [Gas+22].

to take data characteristics into account to steer the user dialogue (e.g., identifying the movie a user wants to see) more efficiently.

For instance, a cinema could have a customer database storing the reservations for movie screenings. A typical transaction to be made accessible using a conversational agent is the ticket booking process, where the users have to specify their `customer_id`, the `screening_id` and the number of tickets. In order to integrate such a task into a typical existing task-oriented dialogue system, we would first have to model the tasks the conversational agent supports (e.g., buy a ticket) along with slots, i.e., the required attributes for the task (e.g., the `screening_id` and `customer_id`).

All this information, however, is typically already available in the given database and the set of its transactions (e.g., implemented as stored procedures or user-defined functions). Therefore, the main idea of *CAT* is to automatically extract and leverage this information instead of asking the user to manually specify it. Moreover, *CAT* then uses this information to synthesize annotated dialogues which are needed to train the conversational agent. Hence, instead of collecting this training data for every domain and database manually, we automate this process. Moreover, the agent and the database are tightly integrated afterwards, and the agent can directly execute the desired transactions without any manual overhead—in contrast to existing task-oriented dialogue systems where a dedicated database integration would have to be developed for every domain.

This tight integration also allows us to use characteristics of the given database (e.g., data statistics) at runtime to guide the dialogue. For instance, to identify the movie a user is interested in, the agent asks the users for properties of the movie (e.g., genre or actors playing in the movie). In the following, we give a brief overview of how *CAT* works as depicted in Figure 3.2:

**Training Data Generation (Offline):** In order to generate a conversational agent, we require training data for both the natural language understanding (NLU) and the dialogue management (DM) models [ZE16]. The NLU model translates user utterances (e.g., ”I want to watch ’Forrest Gump’”)

into annotated slots (`movie_title='Forrest Gump'`) and user intents (ticket reservation). For the NLU training, we generate utterances using a few base templates that are provided by the developer. To form full sentences from these templates, the existing data in the database can be used. In addition, we increase the variety of the natural language by using automated paraphrasing, as we did already for NLIDBs (see Section 3.1.2). Furthermore, to learn typical dialogue flows, i.e., what high-level action to take next (e.g., retry a task after an abort), we generate additional training data using the idea of dialogue self-play [Sha+18], i.e., we simulate different users interacting with a conversational agent. *CAT* then uses this training data to train state-of-the-art models for NLU and DM using the RASA open source conversational AI framework.[7] More details can be found in Section 10.3.

**Data-aware Dialogues (Runtime):** At runtime, the dialogue outlines created in the last step already determine the high-level flow of the dialogue. In addition, the conversational agent has to decide on the low-level flow, to determine which information should be requested next from a user to uniquely identify an entity required for a task, e.g., it could decide to ask for the movie title to identify the movie. In current approaches, this selection is usually done by learned models operating just on the previous input by this user [Sha+18]. In contrast, in order to efficiently narrow down the candidate movies, our approach takes information such as the selectivity of attributes already in the database into account. In addition, we allow adding an annotation to the database schema indicating which of the attributes are probably unknown to the customer. For instance, even though the `screening_id` is very useful and ultimately required for the transaction, the user will most likely not be aware of it and the conversational agent should thus not request it from the user. This results in more succinct dialogues, since the agent quickly gathers the information needed for a transaction.

In particular, the best information (i.e., a so-called slot) to request depends on (i) the probability that the user knows a certain attribute and (ii) how much this attribute narrows down the current set of candidates. Learning both factors end-to-end means learning the database content along with user preferences simultaneously, and again requires a large amount of data. We thus propose a different approach and explicitly keep track of the candidates (e.g., the screenings that match the previous user preferences) and request the next attribute based on the data distribution of the candidates and the likelihood that the user can provide this information. Moreover, while existing task-oriented dialogue systems implicitly assume that the database consists of just a single table [Sha+18], we can seamlessly integrate foreign-key dependencies, e.g., a user can provide information about actors to narrow down the set of possible screenings via the movie relation.

Another advantage of this data-awareness is that no retraining is required in case data changes. The updated database is simply leveraged at runtime to steer the dialogue. More details on the data-aware dialogue steering can be found in Section 10.4.

### 3.3.4. Key Findings

We compared several configurations of *CAT* to state-of-the-art approaches for intent classification and slot filling, using the widely used ATIS spoken conversation corpus [HGD90]. In contrast to all baselines that require manually crafted training data, our approach only relies on synthesized

[7] `https://rasa.com/`

training data, but still reaches comparable performance for slot filling. Moreover, on the intention classification task, it even outperforms multiple baselines.

To evaluate the effectiveness of our data-aware slot selection policy, we compared it to static and random selection strategies using a movie database and again the ATIS dataset. The speedup (in terms of interaction turns) compared to a random strategy can be up to $80\,\%$ for large tables with many dimensions to join. When large amounts of data similar to the production entries are already available at training time, the static strategy can reach a similar performance as our data-aware policy, but will not adapt to data distribution changes at runtime. Additionally, it cannot react to systematic problems in uniquely identifying entries of some tables (caused by data characteristics like almost identical entries). An integrated caching strategy leads to an average response latency of only a few milliseconds.

## 3.4. Discussion & Future Research

Natural language interfaces can simplify the daily routine for professional users who can use their intuitive human language to get required data from a structured source instead of building complex SQL queries. Furthermore, they allow people without computer science (CS) knowledge to access and manipulate structured data. Using natural language in text and speech interfaces may be more accessible for, e.g., elderly or visually impaired people than a graphical user interface and therefore enable them to book a ticket for a train or an event online or at a machine without the need for personal interaction that cannot be provided for demographic or economic reasons [AR22].

As we showed in this chapter, the underlying task of NL2SQL translation is not sufficiently solved yet. This is also underlined by the recent *BIRD* challenge[8] by Li et al. [Li+23]. The benchmark features $95$ big databases and over $10k$ NL-SQL-pairs. It compares the submission accuracy on the test set to a human accuracy on the same data, showing that even the best approaches are currently more than $30$ percentage points below the human performance of $92\,\%$. Moreover, the authors find that zero-shot approaches using state-of-the-art LLMs only achieve an execution accuracy of $40\,\%$ on the benchmark, whereas more complex approaches combining dedicated learned components and prompting *ChatGPT* [e.g., Don+23; Liu+23] can outperform state-of-the-art models for generic NL2SQL translation. This confirms that training data is still needed for improving the quality of NLIDBs. Furthermore, the analysis shows the importance of database values for accurate NL2SQL translation. This is in line with our findings on the advantages of data-aware approaches we sketched above. Our approaches can help to both generate suitable training data and provide a database integration.

The high associated costs for querying a closed model render may render using them useless for scenarios requiring a low overhead. Furthermore, the typical issues (e.g., hallucination, privacy/data protection issues, lack of explainability) of LLMs have to be taken into account. An alternative could be fine-tuning open-source LLMs as suggested by Gao et al. [Gao+23]—which again requires training data.

In addition to the direct usage of LLMs for NL2SQL translation, it makes sense to further evaluate how they can be of help for components other than the translation itself in the future. Possible directions would be the generation of additional templates for weak supervision approaches,

[8] `https://bird-bench.github.io/`

additional paraphrasing of NL utterances, or hybrid approaches where it is possible to directly query a LLM when the results of the low overhead model are not sufficient.

Further improvements to *DBPal* could be made by introducing additional preprocessing steps, e.g., a component that maps between human-readable and abbreviated table names like the one proposed by Zhang et al. [Zha+23].

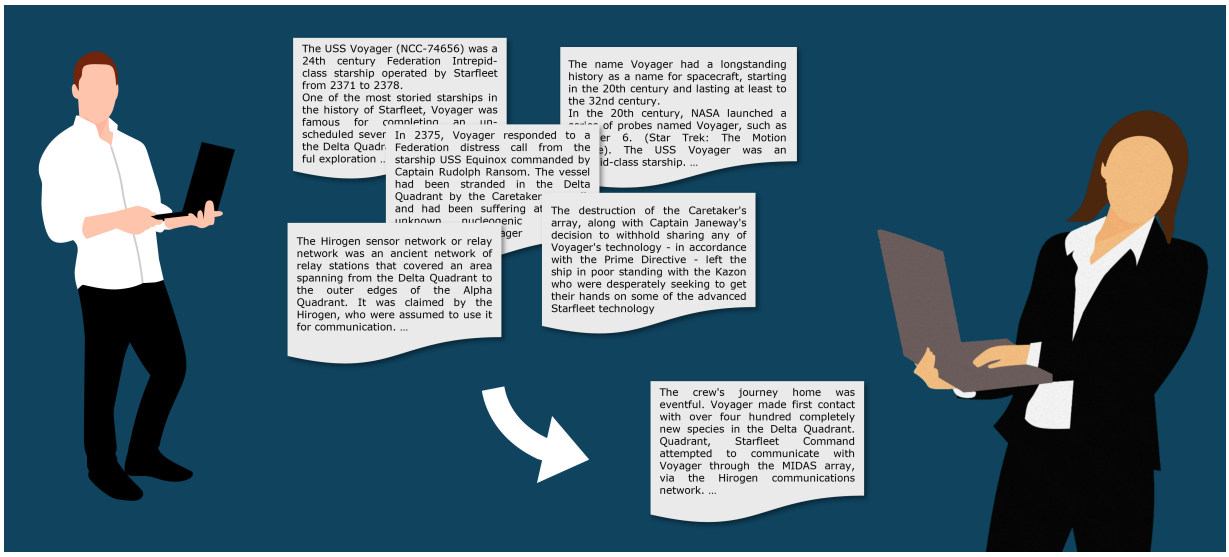
# 4. Personalized Summarizations of Text Collections

In the last chapter, we discussed ways to democratize access when data is nicely structured in a database. But what when this is not the case? When users are confronted with large, unsorted text collections, maybe not even knowing what exact topics they cover?

There already exist systems for text exploration like [Glo+13] and [FG17] that allow data scientists of varying skill levels and even novice users to interactively analyze unstructured text document collections—however, those systems concentrate mainly on keyword searches and document ranking.

While keyword-based search systems are important to filter down the number of relevant documents, they still do not support users in semantically understanding the document collection. Imagine for example a journalist who just received a large collection of documents to start an investigative case, a lawyer who needs to screen a large collection of e-mail conversations, or fiscal authorities trying to detect tax evasion and money laundering in leaks like the *Panama Papers* or the even bigger *Pandora Papers*—with terabytes of data and millions of documents each, processing them is only possible if the tax offices have suitable tools at hand.

In all these examples, an important step to better understand the collection of texts and find the overall relation and event structure of those documents is to produce a concise textual summary that captures most of the important information relevant to a user's individual goal. Yet, we argue that there is not the single information that is most relevant to all users, but each of them will have a different focus and so should the summaries, too.



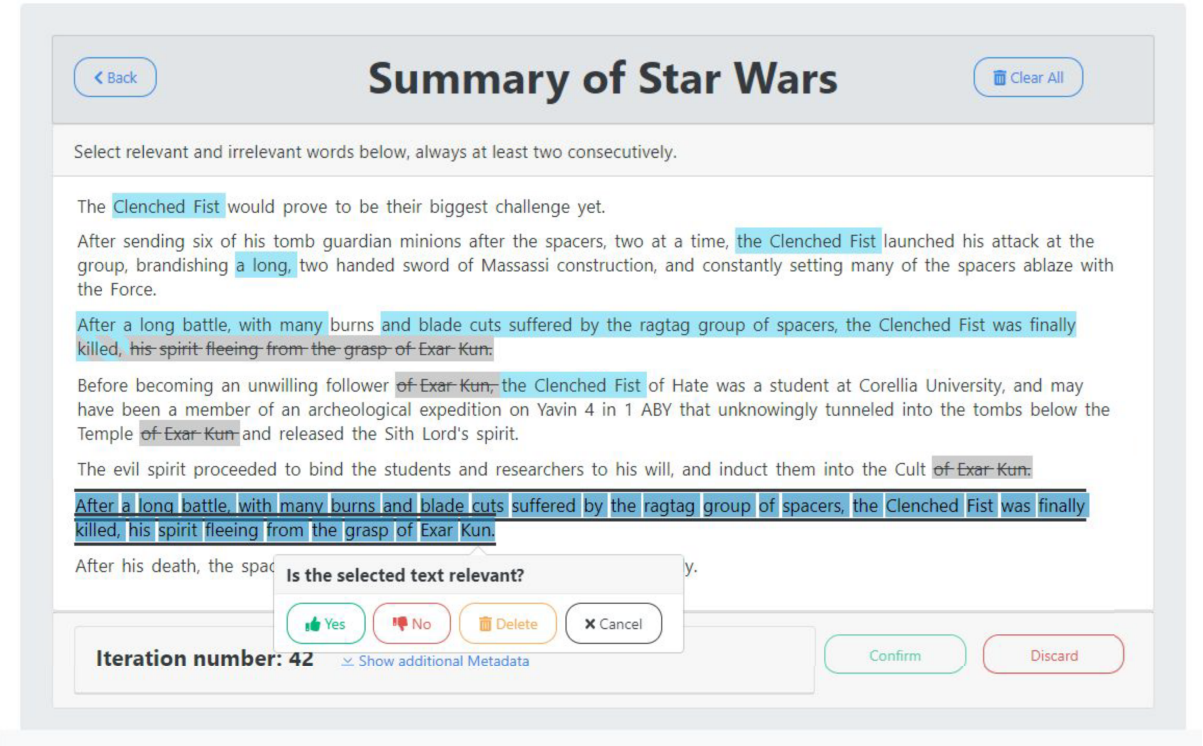



Figure 4.1.: Web-based interface for *Sherlock*, allowing to mark parts of the summary as relevant or irrelevant, and view and adapt annotations from previous iterations.

We therefore present a system to automatically generate summaries with low overhead, that is (mainly) intended for exploration purposes (see Section 4.1). To complement this research, we propose a way to automatically acquire additional corpora for the development and testing of multi-document summary (MDS) systems in Section 4.2.

## 4.1. Personalized Summarization: Sherlock

Users like journalists or lawyers confronted with a large collection of unknown documents need to find the overall relation and event structure of those documents. An important step for this understanding process is to produce a concise textual summary that captures the information most relevant to a user's aims (e.g., degree of details or covered topics). While there is a broad range of automatic text summarization approaches, only few of them produce different summaries targeted at the individual user. Moreover, we want to achieve that adaption to the user with low overhead, i.e., at low costs and without long training times.

An approach for automatic summarization that provides customization through interaction and not by other costly actions like model fine-tuning was proposed by Avinesh P. V. S. and Meyer [AM17]; unfortunately, their approach does not scale for large corpus sizes. We leverage techniques from database research and propose a sampling-based system that builds on their approach which allows a real-world usage to achieve democratization.

**Publications:** The work on *Sherlock* was mainly published in three peer-reviewed publications, of which two are part of this dissertation: We published the vision and general idea in the extended abstract "Benjamin Hättasch. 'Towards Interactive Summarization of Large Document Collections'. In: *Proceedings of the First Biennial Conference on Design of Experimental Search & Information Retrieval Systems, Bertinoro, Italy, August 28-31, 2018*. Volume 2167. CEUR Workshop Proceedings. CEUR-WS.org, 2018" (see Chapter 11). Afterwards, we created a first demo that integrated a basic sampling approach into an existing system by Avinesh P. V. S. and Meyer [AM17] for personalized summarization and published it as "Avinesh P. V. S., Benjamin Hättasch, Orkan Özyurt, Carsten Binnig, and Christian M. Meyer. 'Sherlock: A System for Interactive Summarization of Large Text Collections'. In: *Proc. VLDB Endow.* 11.12 (2018)". The creation of this demo was led by Avinesh P.V.S. and is therefore not part of this dissertation. Finally, we created a full version of the sampling approach and published it as "Benjamin Hättasch, Christian M. Meyer, and Carsten Binnig. 'Interactive Summarization of Large Document Collections'. In: *Proceedings of the Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD 2019, Amsterdam, The Netherlands, July 5, 2019*. ACM, 2019" (see Chapter 12). This includes the careful evaluation of multiple sampling strategies and the development of a cost model for the sampling size.

**Contributions of the author:** I was the lead author of both publications considered here. As mentioned above, the demo application not considered in this thesis was created by Avinesh P.V.S. building upon a prototypical implementation in the context of a master thesis by Orkan Özyurt co-supervised by me and the other authors of the paper. For the main publications, I was responsible for developing and evaluating the sampling approach and writing the manuscripts. The co-authors of the second publication, Christian M. Meyer and Carsten Binnig provided invaluable feedback. All authors agree with the use of the publication for this dissertation.

### 4.1.1. Our Approach

The task of producing textual summaries from a collection of documents is a well-established task in the text analysis community [NM11]. Despite a lot of research in this area, it is still a major challenge to automatically produce summaries that are on par with human-written ones. To a large extent, this is due to the complexity of the task: a good summary must include the most relevant information, omit redundancy and irrelevant information, satisfy a length constraint, and be cohesive. But an even bigger challenge is the high degree of subjectivity in the summarization task, as it can be seen in the small overlap of what is considered important by different users [AM17]. Optimizing a system towards one single best summary that fits all users, as it is assumed by current state-of-the-art systems, is highly impractical and diminishes the usefulness of a system for real-world use cases.

Avinesh P. V. S. and Meyer [AM17] have shown that user feedback significantly improves the quality of the summary. However, each iteration of taking user feedback into account to create a new summary can take from several seconds for small document collections to hours for larger collections, since the runtime will grow exponentially with the input length, as shown in Figure 4.2 (black line). Since the customization of the summary depends on the user's feedback, it is one of the most important aspects to keep users involved in the exploration process. Yet this can hardly be reached with long iteration times of multiple hours, even waiting times of minutes or multiple seconds can already cause the user to lose interest. A previous study [LH14] has shown that even

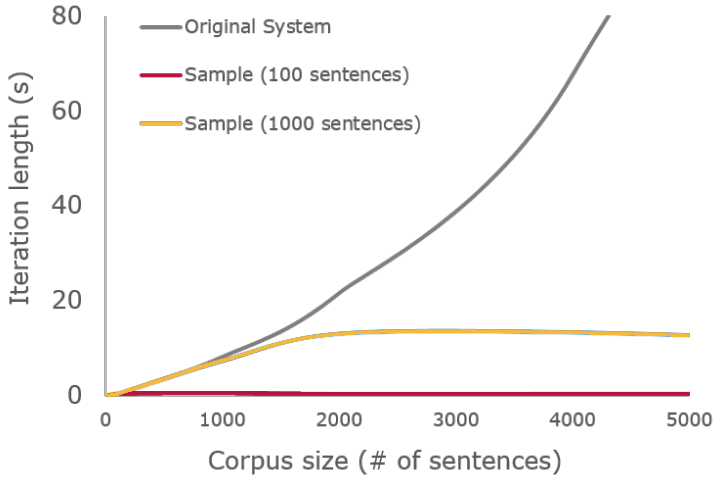



Figure 4.2.: Scalability of Text Summarization Models. Adapted from Avinesh P. V. S. et al. [Avi+18].

small delays of more than 500ms significantly decrease a user's activity level, dataset coverage, and insight discovery rate.

We therefore propose our system *Sherlock* that builds on top of the research of Avinesh P. V. S. and Meyer [AM17]. In order to provide interactive response times in each iteration of the summarization procedure, we are using a novel approximate summarization model. The main idea of this approximate summarization model is similar to approximate query processing in databases: instead of looking at the complete document collection in every iteration, we only consider a sample from the documents per iteration to compute the summary. As a main contribution, we propose a method to select the sample size based on iteration time thresholds and evaluate multiple different sampling strategies. As we show in Figure 4.2, that way our approximate summarization model can provide interactive latency for each interaction loop independent of the size of the text collection that is being summarized (yellow and red line).

*Sherlock* consists of two major components as shown in Figure 4.3: a web-based user interface to collect the user's feedback and a backend that refines the text summarization model. The backend hosts multiple components: a document store (input docs in Figure 4.3) including the required indexes, the summarization component that accumulates the user feedback and learns to create summaries for every iteration as well as our approximate model to execute the summarization process at interactive speeds.

**User Interface:** The web-based interface allows users to summarize a collection of textual documents in an interactive manner. A screenshot of the interface can be seen in Figure 4.1. In a typical setup, a user would need to read all the documents and manually summarize them. In our interactive setup, the user receives a summary, annotates all important and unimportant (parts of) sentences, and submits them as feedback for the next iteration where a refined summary is created by *Sherlock* and the user provides the next round of feedback.

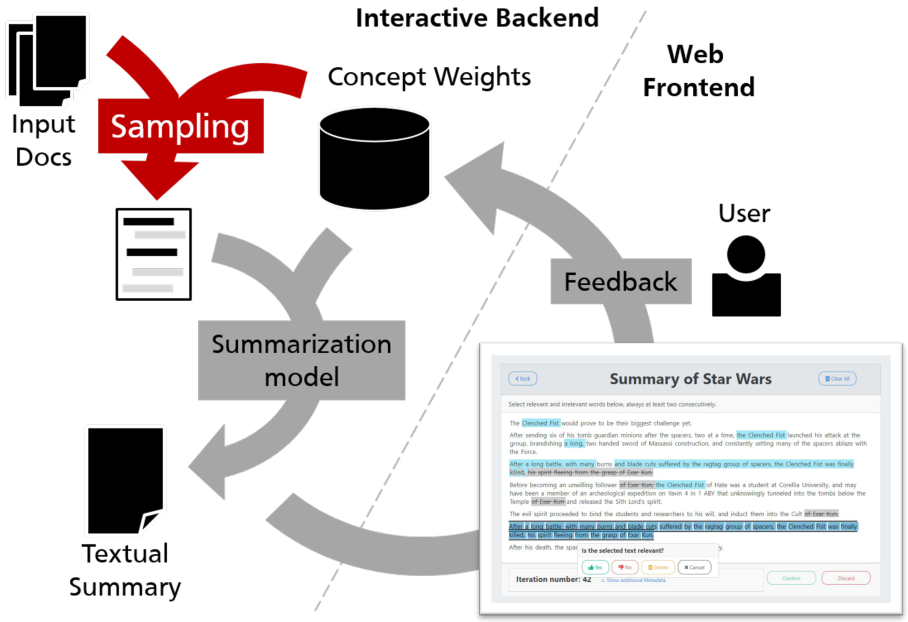



Figure 4.3.: System Overview of *Sherlock*. Adapted from Hättasch, Meyer, and Binnig [HMB19].

**Interactive Backend:** The main task of the backend is to compute the summary for each iteration by taking the documents and the user feedback into account. In our system, we currently employ the summarization model as presented in [AM17] which maximizes the weighted occurrence of concepts in the summary by using an integer linear program (ILP). The first summary which is presented to the user is based on a model without any user feedback. Afterwards, in every iteration the summarization model is refined based on the user feedback of all previous iterations; i.e., the user can adjust the concept weights and hence the ILP needs to be re-executed. Instead of using the full document corpus as input, our backend uses samples of a given size, which will influence runtime and expected quality.

### 4.1.2. The Approximate Summarization Model

The main idea of our approximate summarization model is to take the user feedback of the last iteration into account, adjust the summarization model, and then return a new version of the summary to the user. As discussed before, in order to achieve interactive response times in every iteration, the approximate summarization model takes a sample of the overall document collection as input. The sampling strategy and the sample size have a big impact on the performance and the quality of the summarization model. More details can be found in Section 12.3.

**Sampling Strategy:** We suggest an importance-based strategy (called `TOP-K` as well), that takes the importance of a sentence into account when sampling from the underlying document collection (i.e., more important sentences are sampled with a higher likelihood). Our intuition is that sentences with a higher *information density* (containing more concepts rated as important) are more relevant to the user. As concepts, we use bigrams as suggested by Avinesh P. V. S. and Meyer [AM17]. We initialize the weight of a concept using the *document frequency*; i.e., the number

of documents in the collection the concept appears in. The information density of a sentence is the average weight of all concepts in the sentence. Based on the user feedback, we increase and decrease the weights of the concepts, yielding refined information density scores. In every iteration, we induce a sentence ranking and select only the top-$k$ sentences based on the information density. The strategy can be further extended by first clustering the input sentences based on sentence embeddings [Con+17], and then independently sampling from these clusters to create more widely-focused summaries. We call this variant `STRATIFIED`. The sampling strategy introduces some computation overhead but allows exploitation of collected feedback already in the sampling process. Other evaluated strategies can be found in Section 12.3.

**Sample Size Estimation:** Our sampling strategy requires choosing a sample size. This parameter should not be selected arbitrarily, since a small sample might not contain relevant sentences, but a big one will increase the runtime and might cause the user to stop giving feedback before the desired quality level is reached.

At the core of our system, as discussed before, an ILP solver is used that maximizes the accumulated weight of all concepts in the summary for a given summary length (i.e., consists of sentences mainly containing the highest-rated distinct concepts based on the user feedback). In order to set the sample size, we use a *cost model* to estimate the response time of the system. The main intuition behind our cost model is that each sentence in the input to the ILP produces additional constraints that have to be respected for finding the summary in the next iteration by the solver. Fewer constraints make it easier for the solver to find a solution and therefore reduce the computation time. The cost function thus only depends on the sample size (which directly translates into the number of constraints) but not the summary length, since this is only present as a single constraint in the ILP. Hence, different summary lengths with the same amount of input concepts will still yield similar runtimes of the summarization system.

### 4.1.3. Key Findings

First, our experiments (see Section 12.4.1) confirm that our cost function closely resembles the actual runtime of the ILP solver. With just a few calibration runs, it can therefore be used to derive the maximum sample size $k$ such that the runtime stays below a chosen interactivity threshold (e.g., $500\,\text{ms}$).

As discussed before, the sample size should not be purely selected based on the estimated runtime, but should be big enough to allow for a certain quality of the result. We therefore compared the quality (i.e., similarity to (human-written) gold summaries) of our system and the original one for a wide range of artificial and human-written datasets, using different sample sizes and amounts of feedback (see Section 12.4.1 for the details). As a result, we see that the quality of summaries produced with only $10\,\%$ sample size is nearly the same as for the full data and the mean quality of the system working on a quarter of the input data is indistinguishable from the mean quality of the original system while the length of each iteration is only about $20\,\%$ of the one from the original system. This knowledge can be used to guide a user that is asked to specify a computation budget.

Additionally, one should make sure the sample contains enough sentences to fill the full summary and the summarizing model is still able to choose from different sentences, even if that requires slightly more runtime than advised by the cost function.

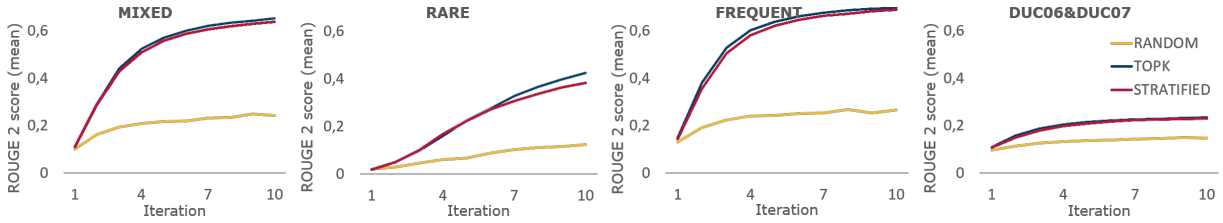



Figure 4.4.: Mean ROUGE2 scores for running the system for 10 iterations with our two sampling strategy variants compared to random sampling as baseline on our summaries (MIXED, RARE and FREQUENT from left to right) as well as real data (rightmost plot, original DUC06 & DUC07). 380 different summaries were used per plot. ROUGE2 scores are always between 0 and 1, higher scores are better. Adapted from Hättasch, Meyer, and Binnig [HMB19].

Finally, we want to show that our complex sampling strategies provide an advantage over trivial approaches like random sampling. To do this, we also considered different kinds of summaries. All details can be found in Section 12.4.2.

There are two important dimensions of textual summaries that have a major impact on how well a sampling strategy works: The first dimension is which concepts are included in a summary (i.e., frequent ones or rare ones). The second dimension is whether the summary is topically focused (for users who are interested in particular details) or if it contains a wide range of concepts (for users who want to get an initial overview).

In order to evaluate our sampling strategies on these different summary types, we artificially created different summaries that follow the above-mentioned properties from the existing DUC06 [DUC06] and DUC07 [DUC07] corpora. That way, we can control the content of the summaries and make sure they still have the syntactic properties and word distributions of a real text. We create three categories: summaries with concepts that appear frequently (called FREQUENT), summaries with only concepts that are rare (called RARE), and ones with a random mix of rare and frequent concepts (called MIXED). Furthermore, we use the summaries included in the original DUC corpora. All gold summaries have a length of about 250 words.

As we show in Figure 4.4, both variants of our sampling strategy clearly outperform random sampling; thus both can be used to generate high-quality summaries in only a few iterations. Moreover, as expected, both strategies work better on the summaries of types MIXED and FREQUENT and are less effective on summaries of type RARE since the sampling strategies prefer more important concepts over less important ones.

Our system also works reasonably well on the original summaries from the DUC06 and DUC07 corpora, but the scores are much lower on average (see right-most plot of Figure 4.4). Yet this is not caused by the algorithm or sampling strategies, but by the fact that the gold summaries of the *DUC* corpora are abstractive and not extractive ones (i.e., summaries are not composed of full sentences from input documents but newly written using other words). This is in contrast to the summaries used for the left three plots of which are extractive, leading to a higher possible overlap between the reference summary and the summary produced by our system.

Our experiments show that our system can indeed reach a speedup compared to the original one that allows for interactive usage. Furthermore, the quality of the resulting summaries will not be negatively influenced by sampling.

## 4.2. Evaluation: Fandom Corpora

While working on *Sherlock*, we noticed that there is a lack of high-scaled datasets for evaluation of MDS systems. Existing resources are mostly limited to the news genre [e.g., Her+15; NGV12; PXS18] and a handful of shared tasks [e.g., DUC06]. The news corpora have been used successfully for training a wide range of neural architectures [e.g., GMG18; GDR18; NZZ17; NCL18; PXS18; SLM17], but are limited to texts that are typically too short to qualify as general-purpose summaries. For example, *CNN/DailyMail* provides only bullet-point summaries, *Gigaword* contains headlines as the summary of an article's first sentence, and the *NYT corpus* pairs news articles with their abstracts. To break new ground in the automatic summarization of other genres, we require new corpora that can cover other text genres and summary types on the one side, but are large enough to train neural networks on the other side. However, manually creating corpora is hard, since they require human-written (gold) summaries of text at a large scale. Acquiring them is expensive and additionally requires extensive quality assurance measures, especially when crowd-workers are employed for that. Furthermore, the desired properties (e.g., the size, or having summaries for a wide range of topics vs. a single domain) might differ across applications. Thus, to accompany our research on (personalized) summarizations, we created a novel approach to automatically build summarization corpora from existing text collections.

**Publication:** We published this work as "Benjamin Hättasch, Nadja Geisler, Christian M. Meyer, and Carsten Binnig. 'Summarization Beyond News: The Automatically Acquired Fandom Corpora'. In: *Proceedings of The 12th Language Resources and Evaluation Conference, LREC 2020, Marseille, France, May 11-16, 2020*. European Language Resources Association, 2020". In this paper, we present a novel automatic corpus construction approach to tackle issues mentioned before, as well as three new large open-licensed summarization corpora based on our approach that can be used for training abstractive summarization models. We provide a ready-to-use framework that implements our automatic construction approach to create custom corpora with desired parameters like the length of the target summary and the number of source documents from which to create the summary. The main idea behind our automatic construction approach is to use existing large text collections (e.g., thematic wikis) and automatically classify whether the texts can be used as (query-focused) multi-document summaries and align them with potential source texts. We show the usefulness of our automatic construction approach by running state-of-the-art summarizers on the corpora and through a manual evaluation with human annotators.

**Contributions of the author:** I was the lead author of the publication and thus responsible for developing and implementing the approach, conducting the experiments, and writing and submitting the manuscript. The co-authors Nadja Geisler, Christian M. Meyer and Carsten Binnig contributed invaluable feedback. All authors agree with the use of the publication for this dissertation.

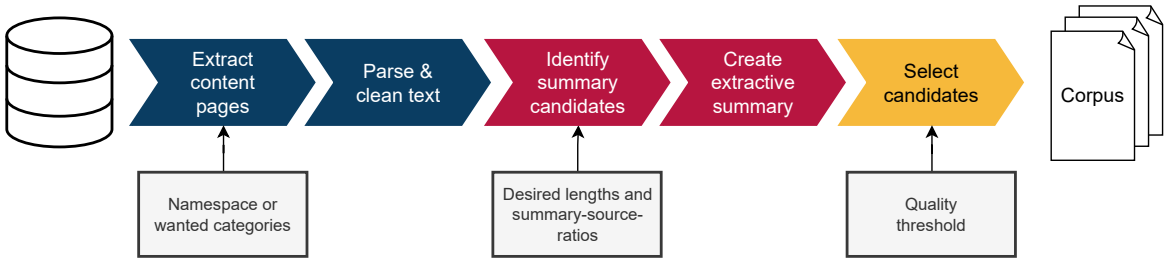



Figure 4.5.: Fandom Corpus Construction Pipeline. Adapted from Hättasch et al. [Hät+20a].

### 4.2.1. Our Approach

The central idea of our approach is using existing large text collections (e.g., thematic wikis), automatically classifying whether texts can be used as (query-focused) multi-document summaries, and then aligning them with potential source texts.

As an important first step for developing such an automatic construction approach, we use the Fandom wikis (formerly known as *wikia*). They currently consist of more than 385,000 communities on different franchises (movies, television series, games, books, and more) with over 50 million pages in total. The sizes of the different communities range from only a few pages to well over 100,000 content pages. Most of those wikis use an open *Creative Commons Attribution Share-Alike* license, allowing us to use and redistribute their articles.

The Fandom wikis often contain articles describing the same topic in multiple levels of detail—there are articles giving a general overview of a character, event or place as well as articles focusing on a single aspect of it (e.g., a relationship, scene or time) in detail. Those articles normally reference each other through links. Our main idea is to automatically identify such overview articles or sections that qualify as a summary and align them with the potential source documents (i.e., the detailed articles) if the supposed alignment quality is high enough.

This allows to generate multiple different corpora with user-defined properties. For example, it is possible to vary the target length of the summaries, but also the difficulty of the summarization task, which we control by the ratio between the sizes of summary and source documents. The corpora can be constructed based on a single community or cover a broader range of topics by merging contents from multiple wikis.

The essential stages of our approach to create topic-specific multi-document summarization corpora are: (1) parsing and cleaning of input documents, (2) selecting potential candidates for abstractive summaries from those input documents and assigning summary candidates to them, and (3) choosing the final set of abstractive summaries based upon a newly developed quality threshold and splitting the selected summaries into training, validation, and test set if needed. An overview can be found in Figure 4.5.

In the first step, wiki dumps are processed and non-content pages are automatically discarded based on metadata. Afterwards, the contents are preprocessed: We split the pages into sections including their respective titles, extract the links between pages and convert the content into plain text. This includes removing link texts, tables, templates and other kinds of wiki markup. The second and third step, which are the core of our automatic construction approach, are implemented in a general way, to allow transferring it to sources different from (fandom) wikis, too. We will

now give an overview on the underlying ideas of the two core steps, more details on them as well as the first step can be found in Section 13.2.

**Finding Summary Candidates:** The identification of summary candidates is the most crucial step for creating high-quality corpora automatically. At a high-level, a corpus that is useful for abstractive summarization should group a set of documents with at least one possible summary of these documents. In addition, many of the automatic summarization approaches take a "query" as input that represents the information needs of the user, i.e., describes what aspects are important.

Hence, in this step, we aim to select triples (i.e., a set of source documents, a summary, and a query) that represent good candidates from a given cleaned data dump for the final corpus. For both the *source documents* and *summaries* our pipeline uses sections of the wiki articles since they are coherent and self-contained portions of text. As a *query* describing the section, we combine the title of an article with the section title, e.g., "Luke Skywalker: Birth".

As a pre-filtering step to identify sections that qualify as possible summary candidate triples, we use the following heuristics:

1. Only sections with a length between certain threshold values are considered as summaries. These thresholds can be adapted based on the task at hand. More details on the parameter selection can be found in Section 13.2, where we also list the parameters used to create the three sample corpora that we published.

2. We discard summary candidates having only few linked documents (i.e., potential source documents). Again, the number of source documents is a user parameter that will influence the difficulty of the summarization task.

3. After applying these purely statistical heuristics, we compute the content alignment between summary and source documents as the overlap between sources and summary candidates. On that score, a threshold will be applied; the lower the value, the more candidate summaries and source documents will be selected but the difficulty increases. We use the number of shared bigrams to approximate the similarity. This overlap shows how much the summary and source texts contain similar concepts, but it can only be a first hint whether the information in the sources is sufficient to re-create the abstractive summary given a particular user-query.

This step will result in a much shorter list of candidates for triples, but the quality still varies drastically: for some of them, the summary is indeed a high-quality summary of the extracted documents complying with the query, while for others it is hardly possible to find the information of the summary in the source documents. Hence, in a final step, we need to identify the usefulness of each triple and select only those which exceed a predefined quality threshold.

**Choosing the final set of summaries:** To check whether the information in the sources is sufficient to re-create a given abstractive summary, we try to reconstruct its contents by extracting matching snippets from the source documents. This can be done automatically and will result in a quality score for each alignment between summary and source texts. As we will show in Section 4.2.2, this computable score corresponds to a human quality assessment. Only texts with a certain score on this automatic reconstruction approach will be considered a valid summary, and are included in the final corpus.

For the automatic reconstruction, which can be seen as extractive summarization with a known target, we use the base idea from our summarization approach *Sherlock* as presented in Section 4.1. The procedure is modelled as an ILP. The main intuition is that the ILP extracts the sentences with the most important concepts from the source documents to form a summary within a maximal length. To model the importance of sentences, we weight concepts according to their frequency in the human-written text (i.e., the selected candidate summary from the Fandom wiki). By doing so, we reward the system for a summary that contains many concepts of the abstractive reference summary. We use bigrams as concepts and ignore those consisting solely of stopwords. The ILP will receive a high score when the extractive summary contains many concepts from the reference summary, and hence it is possible to construct a similar summary from the source texts.

In our publication, we used two different optimization objectives for the ILP. Both approaches use only syntactical features and no semantic ones (e.g., embeddings). They do not require time-intensive training and can be computed within a few seconds. We will print one variant here, and refer for details and the other variant to Section 13.2):

$$
\begin{aligned}
\max \sum_i w_i c_i \\
\forall j. \sum_j l_j s_j \leq L \\
\forall i, j. s_j Occ_{ij} \leq c_i \\
\forall i. \sum_j s_j Occ_{ij} \geq c_i \\
\forall i. c_i \in \{0, 1\} \\
\forall j. s_j \in \{0, 1\}
\end{aligned}
$$

where $c_i$ refers to the individual concepts and $L$ to the maximal summary length (which we set according to the selected range of the target length for the abstractive summaries). Moreover, sentences are referred to as $s_j$ with length $l_j$ and $Occ_{ij}$ meaning that concept $c_i$ occurs in that sentence.

This ILP formulation intends to maximize the overall sum of weights for distinct covered concepts, while making sure that the total length of all selected sentences stays below a given threshold and the weight of a concept is only counted if it is part of a selected sentence.

With both variants, it is possible to create high-quality corpora, as we will now show.

### 4.2.2. Analysis of the Corpora Created with our Framework

#### Properties

In the previous section, we have presented our new approach for automatically constructing summarization corpora. Using this approach, we have created three different sample corpora (one for Harry Potter, two for Star Wars) using the Fandom wikis as input. The resulting corpora have

unique properties of these corpora which differentiate them from other available corpora and, thus, are a valuable contribution on their own.

First, our corpora do not feature news texts with their typical peculiarities (e.g., all important sentences at the beginning) but a mix of encyclopedic and narrative (story-telling) texts. In contrast to other sources, in Fandom wikis there are not a few dozens but thousands of articles about a certain topic. If the corpus is constructed from a single community, all articles are from the same domain (i.e., a closed world). However, it is also possible to utilize the common structure of the different communities and build a corpus containing texts of different domains, e.g., to train more general summarizers.

Additionally, new corpora are fast and cheap to construct, with just a minimum of manual work needed. There are many communities with (ten)thousands of articles, and the wikis are still growing. Moreover, communities are available in many different languages, hence this approach can be used to create corpora for various languages (e.g., one of our sample corpora is in German). The Creative Commons License of the texts allows us to offer the resulting corpora for download instead of only publishing tools for re-creating the corpora. This is in contrast to many existing news-based corpora, which depend on crawling and thus the availability of external resources.

Last but not least, the abstractive texts in our corpora are of high quality since they are written by volunteers with intrinsic motivation and not by poorly paid crowd workers rushing through the task.

### Statistics

Detailed statistics of the created corpora can be found in Section 13.4. The sizes of the summaries match traditional multi-document summarization corpora like the *DUC* challenges. The average number of source documents per summary lies between 19 and 25 documents, again similar to existing corpora. The average length of the source documents varies between the communities and thus corpora, influencing the hardness of the task. The size of the final corpus varies depending on the size of the Fandom community and the quality threshold. For our sample corpora, it ranges from 250 topics, which is similar to the *DUC'06* dataset used for traditional summarization approaches, to 1,300 topics, which is a size that can be used to train deep learning approaches.

Additionally, it is possible to combine topics from multiple communities into a single training corpus. This has an effect on the domain distribution and topic heterogeneity as well. A corpus constructed from a single community covers topics from only one domain, with the main difference between documents being whether they are about an event, a place, a being, or a thing. Mixed corpora may contain texts from totally different domains (e.g., about a movie, a video game and baking recipes). The heterogeneity of writing styles, levels of detail, narrating styles and more, comes from the nature of the wiki itself and is inherently contained in all the corpora.

In summary, it can be seen that, from a statistical perspective, it is possible to generate corpora with various properties matching typical needs of current (multi-)document summarization tasks.

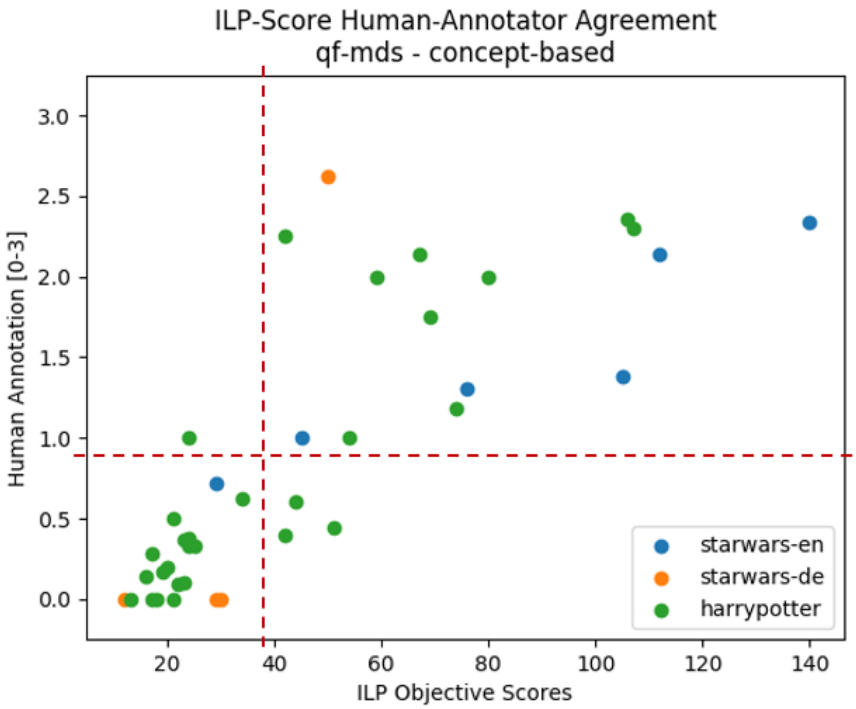



Figure 4.6.: Human agreement with automatically generated extractive summaries for concept-based creation method. Average values for the sentences of 39 annotated documents, possible values between 0 and 3 (best). Published in Hättasch et al. [Hät+20a].

### Quality Evaluation

Besides some task-depending parameters like the target length of the summaries, the size of the corpus on the one hand and its quality on the other hand will be strongly influenced by the threshold applied to the ILP objective score for selecting the final set of summaries. This holds for both methods, since there is a correlation—summaries with a high score on one method will achieve a high (but slightly different) score with the other ILP as well, as we show in Section 13.4.

The correlation justifies using either of the two methods as a quality indicator. However, the question of how the quality of the summary really correlates to the score of the extraction remains. To assess this, we asked human annotators to evaluate the quality of 39 equally distributed summaries. We asked them to decide for each sentence in the human abstract if it is covered by the extractive summary (0) not at all, (1) partially, (2) mostly, or (3) fully. The human decision is averaged for the full summary and correlated to the score of the extraction. The results can be found in Figure 4.6 for the concept-based approach. For all details of the experiment and the evaluation of the second approach, see Section 13.4. There you will also find a sample of a human abstract and the corresponding extractive summary, to intuitively demonstrate that extractive summaries are a good proxy to judge the quality of source documents for abstractive summarization. It can be seen that a higher ILP score does indeed correlate with a better human evaluation. Based on this, we have chosen the ILP-thresholds for the selection of the summaries.

As the final step of our evaluation, we now want to show the quality of the alignment of a summary and its source texts (i.e., how well the summary can not only be reconstructed from the source texts

| Systems | Harry Potter | | Star Wars (en) | | Star Wars (de) | |
|---|---|---|---|---|---|---|
| | Candidates | Selected | Candidates | Selected | Candidates | Selected |
| **Luhn** | 0.0308 | 0.0365 | 0.0523 | 0.0560 | 0.0357 | 0.0412 |
| **LexRank** | 0.0729 | **0.0881** | 0.1049 | **0.1083** | 0.0784 | **0.0849** |
| **LSA** | 0.0421 | 0.0545 | 0.0533 | 0.0584 | 0.0512 | 0.0624 |
| **KL** | 0.0528 | 0.0655 | 0.0808 | 0.0780 | 0.0524 | 0.0617 |
| **ICSI** | 0.0360 | 0.0419 | 0.0423 | 0.0496 | 0.0353 | 0.0412 |
| **UB1** | 0.1744 | 0.2885 | 0.2341 | 0.4115 | 0.3354 | 0.5811 |
| **UB2** | 0.2609 | **0.3746** | 0.3050 | **0.4726** | 0.3847 | **0.6164** |

Table 4.1.: Average ROUGE-2 values for different baselines on all candidates as well as the subset of summaries selected in the final step with the sentence-based approach and a minimum objective score of 50 for each of the three sample corpora. Values between 0 and 1, higher is better.

but really summarizes them with regard to a query). We use several baseline summarizers, namely *TF*IDF* [Luh58], *LexRank* [ER04], *LSA* [SJ04], *KL-Greedy* [HV09], and *ICSI* [GF09] to evaluate this. Additionally, we apply the best scoring model combination for extractive summarization by Kedzie, McKeown, and Daumé III [KMD18], a combination of a *Seq2Seq* model as extractor and an Averaging Encoder. A short description of all these approaches can be found in Section 13.4. The goal is to show that the quality of the automatically created corpora is high enough that state-of-the art summarizers can perform reasonably well on those corpora—but that the task is on the other hand not too trivial, thus the corpora will be of value for training and benchmarking.

For the assessment of summary quality based upon a reference summary, we compute and report the ROUGE metrics. Owczarzak et al. [Owc+12] show that these metrics strongly correlate with human evaluations of this similarity. In addition, to judge the quality of the baselines, we also computed the upper bound that an extractive summarizer could achieve in the best case. An extractive summarization system normally cannot re-create the human-written abstractive text exactly, since the abstractive sentences differ from the sentences of the source texts that can be extracted. Hence, the best overlap between an abstractive and the best extractive text is usually below $100\,\%$. To take this into consideration, we compute and report those upper bounds for extractive systems, as suggested by Peyrard and Eckle-Kohler [PE16] (more details in Section 13.4).

We benchmark all three corpora with both extraction methods and a quality threshold of 50. Table 4.1 shows the benchmark results of all candidates without the last filtering step compared to the results for the selected summaries for the three sample corpora. One can see that the average scores for all systems are higher on the selected summaries, proving that these are, on average, better pairs of summary and source documents. However, in relation to the upper bounds (UB1 and UB2), even the best performing baseline (*LexRank*) can only reach one third to one fifth of the upper bound on ROUGE-2 (for ROUGE-1 and SU4 it is at least half or better).

Training multiple sequence-to-sequence models with the training data from the corpora (see Section 13.4 for detailed results) leads to different results, e.g., depending on the amount and length of the source documents, the linking style of the wiki, and the overall size of the training data set. A combined corpus with summaries from different domains can help to handle the lack

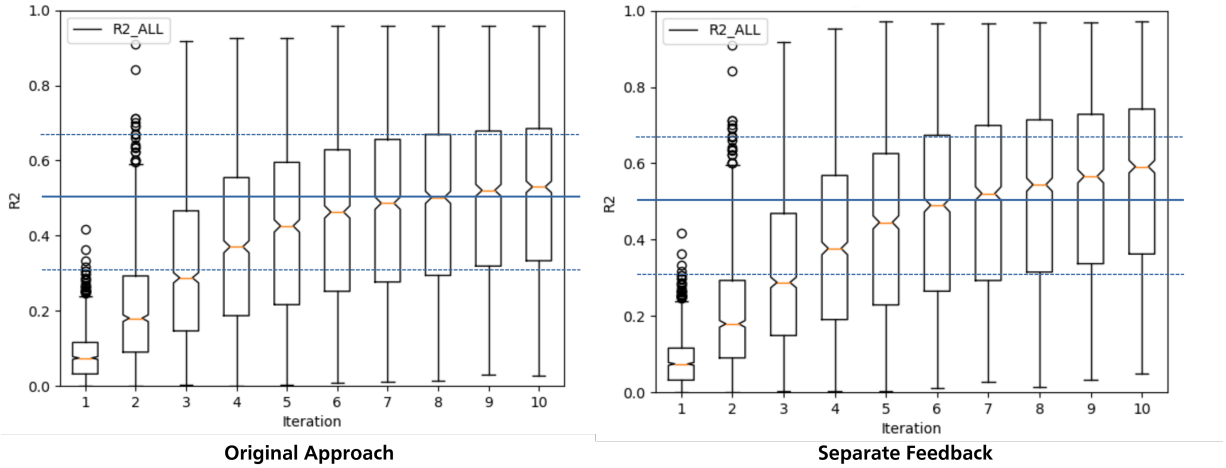



Figure 4.7.: ROUGE2 score distribution for running both our original system and the variant with separate feedback on the datasets described before. The second variant reaches the same average scores with fewer iterations. Scores between 0 and 1, higher is better.

of training data, but we find that a specialized model will outperform this more general model when there is enough training data available.

The results reflect our findings from previous research, e.g., on *Sherlock* (see Section 4.1), and thus we believe that the quality of our automatically constructed corpora is on par with the manually created ones used in previous evaluations. Moreover, the fact that state-of-the-art summarizers can only reach one third to one fifth of the upper bound on ROUGE-2 also emphasizes that multi-document summary is still a challenging task in general and needs further research, which we hope to stimulate with publishing this benchmark approach.

## 4.3. Discussion & Future Research

Personalized and focused summarizations are an important tool to get an overview of complicated and large topics. A real world example can be found in a pilot project called “Vermerkomat” of the State of Baden-Württemberg, Germany. It prepares customized information collections for politicians and ministry staff such that they can take informed decisions [Het23].

Our experiments show that Sherlock can be used for such personalized extractive summarization. A clearly visible adaptation to the user’s focus is reached with merely a handful of rounds of feedback (5-10), even though our system works without a learned model and hence without training data, and with execution times of only seconds on a CPU. However, one can see that the quality usually does not substantially improve further after some iterations. We find that this is often caused by the system converging to a local optimum; this happens when the ILP solver only incorporates sentences into the summary it already got (positive) feedback on. The user can then not give new feedback, and further iterations will always produce the same summary.

As follow-up work to the state published, we therefore started testing a separation of the selection of sentences for the summary and for feedback. First evaluation results are promising: We displayed

additional sentences for feedback in place of sentences where more than half of the concepts already received feedback. As shown in Figure 4.7, this leads to better quality in shorter time. The average quality that was before reached after 10 iterations is now already achieved with 8 rounds of feedback. Afterwards, the quality improves further. In the future, we want to investigate that more thoroughly, and, inter alia, evaluate different strategies to select the sentences for feedback.

Another interesting direction would be extending the feedback process with language models: by representing concepts additionally in an embedding space, the system could automatically distribute weights to unseen concepts based upon the distance in the embedding space, and thus profit from additional information with the same amount of user interaction.

Finally, one could create a hybrid approach where our approach is still used to create an extractive summary, but this is then refined to an abstractive summary using an large language model (LLM) as suggested by Ghadimi and Beigy [GB22]. This would combine the advantages of both directions: the efficiency and low costs of our system is leveraged, and the costs for the LLM prediction stay low, too, since it only needs to process the summary and not all source documents. The LLM could then perform semantic deduplication such that our system can incorporate more concepts in its summary without increasing the amount of text the user has to read.

For evaluating summarization approaches, recent advantages for better text comparison like *ROUGE-SEM* [Zha+24] might be useful. They might help to evaluate the performance of summarization systems more realistically and therefore lead to more targeted enhancements.

# 5. Information Extraction & Integration

The quality of decisions is directly influenced by the available data and the ease of access to it. Important decisions should thus be made based on a holistic view using all available data. However, data is often scattered across different heterogeneous sources and may additionally not be in a format that allows to draw direct conclusions from it. Textual representations might be useful to get an overview, but many decisions require quantification and structured aggregation. It might be needed to compute results based on scattered information to create value and derive knowledge from it. This requires structured representations of the information, e.g., in tabular form.

However, for many tasks, there is no existing database that contains all required information. Source data might exist in structured form, but not yet at the same place, e.g., distributed in separate databases of two formerly separate companies that are now merging. Or it might not even be present in structured representation yet, but only as text. In this part of our research, we will therefore develop approaches to automatically integrate or extract information into structured representations. Deriving these structured representations customized to the user's (information) needs can be the first required step for making wise decisions—either for a task itself or to choose how to continue explorations and investigations.

## 5.1. Applying Embeddings to Information Integration: It's AI Match

As a first step towards customized representations, we therefore investigated how existing relational databases with different schemata can be integrated with low overhead using pre-trained embeddings. This is not only relevant for many data science projects that need to combine data from different independent sources, but also within companies, where data typically resides in different systems. Data Integration can help to mitigate these issues since it allows creating a global view over independent data sources. To accomplish that, semantic correspondences between elements of these different schemata have to be found.

**Publication:** We published this work as "Benjamin Hättasch, Michael Truong-Ngoc, Andreas Schmidt, and Carsten Binnig. 'It's AI Match: A Two-Step Approach for Schema Matching Using Embeddings'. In: *2nd International Workshop on Applied AI for Database Systems and Applications (AIDB20). In conjunction with the 46th International Conference on Very Large Data Bases, Virtual, August 31 - September 4, 2020.* Virtual, 2020" (see Chapter 14). In this paper, we propose a novel end-to-end approach for schema matching based on neural embeddings. The main idea is to use a two-step approach consisting of a table matching step followed by an attribute matching step. In both steps, we use embeddings on different levels—either representing the whole table or single attributes. Our results show that our approach is able to determine correspondences in a robust and reliable way and—compared to traditional schema matching approaches—can find non-trivial correspondences.

**Contributions of the author:** I was the lead author of the publication. A first prototypic implementation of the approach was implemented by Michael Truong-Ngoc in the context of his master thesis, co-supervised by the other authors and me. My tasks consisted of refining the final approach, selecting the experimental design of the publication, and conducting or confirming the necessary experiments, literature research, and writing the manuscript. The co-authors Michael Truong-Ngoc, Andreas Schmidt, and Carsten Binnig contributed invaluable feedback. All authors agree with the use of the publication for this dissertation.

A critical step in data integration is schema matching, which aims to find semantic correspondences between elements of two schemata. Traditionally, this was done by experts with a good understanding of the semantics of the data [Hul97]. However, modern schemata are becoming larger and more complex, making manual schema matching both more time-consuming and more error-prone [RB01]. In order to reduce the manual effort involved in schema matching, many solutions for the automatic determination of schema correspondences have already been developed [Che+12; DR02; Geo05; LN07; MIA17; NHN19; PKT09; Pin+15; RB01; SMJ19].

A major problem of many automatic schema matching approaches is, however, that they fail if the semantic similarity is hard to detect. For example, instance-based column matchers typically fail to match columns that contain disjoint but semantically similar values such two tables with different street names or even worse the same content in different languages (e.g., French and English). Another example are name-based matchers that rely on sources such as *WordNet* to identify column matches: while these approaches can detect hard-to-match cases (such as columns that use synonyms as names), they fail if this knowledge is not encoded in the resource. A detailed analysis of existing approaches and their shortcomings can be found in Section 14.2.

### 5.1.1. Our Approach

To bridge the semantic gap when integrating data from different sources, we propose a novel end-to-end approach for schema matching based on neural embeddings (see Section 14.3 for explanations of how embeddings can be constructed and used for measuring similarity). While neural embeddings have already been used for individual tasks of schema matching (i.e., table or attribute matching), we suggest a new holistic approach that uses neural embeddings on different levels to combine table and attribute matching. To be more precise, in a first step, we use neural embeddings to match the elements of a schema on the table level. For each table in the target

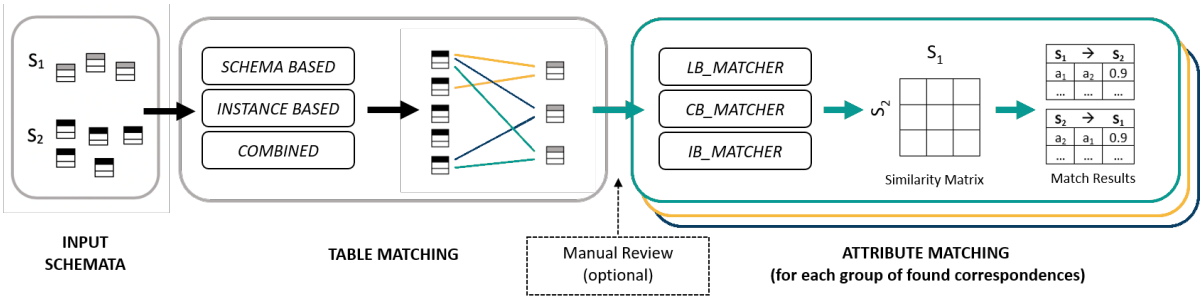



Figure 5.1.: End-to-end process for schema matching, consisting of two steps (table matching and attribute matching). In the first step, possible matches between the tables are produced either on the basis of the available schema information (`SCHEMA BASED`), or on the basis of the instances (`INSTANCE BASED`) of tables or using both together (`COMBINED`). Optionally, after the first step, a user can confirm or reject possible table matches. For the next step, matches between the individual attributes of the remaining table pairs are determined. This can be done using table and attribute names (`NB_MATCHER`), natural language comments in the database schema (`CB_MATCHER`), or instances of a table (`IB_MATCHER`). Either thresholding or ranking can be applied to the resulting similarity values to determine the final matches. Adapted from Hättasch et al. [Hät+20b].

schema, we propose either all tables from the source schema where the similarity is above a certain threshold or the $n$ matches with the highest similarity. In the second step, the system then determines suggestions as to which attributes from the corresponding tables fit together. In addition, after the first step, a user can optionally review and select those table pairs that should be kept for the second step. This holistic approach allows our approach to find non-trivial correspondences such as those discussed before. An overview of our approach can be seen in Figure 5.1, we will now give a short overview of the two steps proposed.

**Table Matching:** Two general approaches are possible here: First, the schema information can be used in the form of table name and attribute names. Tables with semantically similar schema information are probably used to store similar content. Second, exactly these contents of the tables can be examined: we can use the contents of each table to compute embeddings for the tables (either based on all attributes or a subset of it). These embeddings can then be compared to other embeddings to find possible (partial) table matches. Since the candidate pool for this comparison is a cross product of the attributes of all tables which we wanted to avoid with the two-step approach and the calculation of the data embeddings can be expensive to compute, a combined approach is advisable: first the number of table pairs to be examined is reduced schema-based, and then the candidate pool is further reduced using an instance-based approach. For optimization, some intermediate results of the table matching can be stored for the attribute matching step following afterwards. More details on our table matching procedure can be found in Section 14.5.

**Attribute Matching:** In this step, we only use the table candidates that qualify based on table matching, as discussed before. For the attribute matching, we can again use schema or instance information, analogous to table matching. If no further auxiliary information (e.g., strict type

annotations) is available, all attributes of a source table must be compared with those of the target table. We again propose two different types of matchers: Name-based matching uses the table information such as table name and attribute names, as well as available comments. Instance-based matchers inspect the contents of the columns to find attributes with semantically equivalent instances. We explain our attribute matching procedure in further detail in Section 14.6.

### 5.1.2. Matching Strategies & Key Findings

Gromann and Declerck [GD18] have shown in their investigations that for aligning ontologies better results can be achieved with word embeddings than with traditional string comparison methods. We wanted to investigate whether this also holds for matching relational databases. Furthermore, we examine the use of embeddings that can dynamically generate vectors for each input, thus avoiding out-of-vocabulary errors and not requiring explicit handling of multi-word expressions.

We conducted an extensive series of experiments for different strategies and combinations of strategies for both steps that can in detail be found in Section 14.5 (table matching) and Section 14.6 (attribute matching). The experiments were carried out on the English *Mondial DB*,[1] the German *Terra DB*,[2] five *XDR* schemes[3] for customer orders (CIDX, Excel, Noris, Paragon and Apertum), the *OAEI* Benchmark,[4] two film datasets,[5] and the *Adult* dataset.[6]

#### Step 1: Table Matching

For computing table similarity using a table embedding that relies on schema information, we represent each table by a vector for the (equally weighted) combination of table name and all attribute names. The representations are determined using the pre-trained Google USE [Cer+18] model. We show that with such multilingual embeddings, cross-lingual table matching is possible, where purely syntactical approaches will mostly fail.

A qualitative analysis (see Section 14.5.1) of pure schema-based table matching shows that the reason for low precision values at lower thresholds is probably due to similar schematic information in the different tables. For example, almost all tables contain the attribute *name* in slightly modified form. Nevertheless, the experiments show that if the attributes are sufficiently unique, it is possible to determine schema-based table correspondences with embeddings.

Alternatively to only using schema information to compute embeddings, we can also use instances to compute table embeddings. Even with well-existing schema information, additional instance information can help to further increase matching accuracy [Do05]. Studies show that traditional methods can even achieve better results with instance-based matching than with name-based matching (depending on the quality of the instances) [LN07].

[1] https://www.dbis.informatik.uni-goettingen.de/Mondial/
[2] https://www.sachsen.schule/~terra2014/index.php
[3] https://dbs.uni-leipzig.de/bdschemamatching
[4] Ontology Alignment Evaluation Initiative. http://oaei.ontologymatching.org/2009/benchmarks/
[5] https://github.com/AhmedSalahBasha/schema-matching/blob/master/imdb.csv and https://github.com/AhmedSalahBasha/schema-matching/blob/master/rotten_tomatoes.csv
[6] http://archive.ics.uci.edu/ml/datasets.html

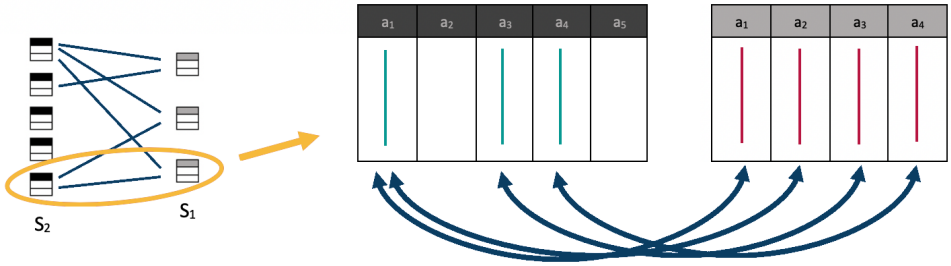



Figure 5.2.: Instance-Based Table Matching – The similarity of tables is inferred from the averaged similarity value for each attribute correspondence. Adapted from Hättasch et al. [Hät+20b].

In instance-based matching, a distinction is made between horizontal and vertical matching [LN07]. Vertical matchers compare column contents of individual attributes and infer attribute correspondences, while horizontal matchers attempt to identify duplicates (i.e. two or more representations of the same object) between two schemata.

We propose an approach for vertical schema matching that uses embeddings to represent the entire content of a column by a single vector and then compares them with each other (see Figure 5.2). Since we aim to evaluate (word) embeddings, we only consider string attributes here. If the resulting vector representations for attributes are sufficiently similar, an attribute correspondence can be assumed.

Such a column vector could naïvely be constructed by averaging the vector representations obtained from a pre-trained model of all instances, analogous to the sentence embedding approach from Arora, Liang, and Ma [ALM16]. However, this approach assumes that all instances are equally important. Analogous to *stopwords* in classical natural language processing (NLP) approaches, there may be instances that contribute less to the semantic meaning, for example placeholders like *not-in-universe*, *unknown*, *NONE* etc. Such noise instances [Dil19] may dominate the representation if no countermeasures are taken. The averaging approach also ignores the order of instances, but these are usually irrelevant to the semantic meaning of an attribute [Geo05].

The aim of our approach is therefore to combine instances to form a single vector representation for attributes that most closely reflects the semantics of the attribute, using frequency values but also relativizing them if necessary (to avoid statistical bias [Mug02]). Sampling based on frequency values can be used for this purpose. At the same time, however, the frequency of an instance is not a clear indicator of whether the information is relevant or not: if not applied carefully, sampling might amplify the noise and thus de-emphasize the representation.

We evaluated three sampling methods:

**Distinct Sampling:** Ignore duplicates when generating the combined representation. This might lead to information loss if the instances are not equally distributed.

**N-Random-Sampling:** Take $n$ random instances, sampled with the same probability over the distribution in the column. this leads mostly to a balanced sample if the sample sets are large enough so that the column is well represented, and is considered the safest way to counteract a statistical bias in the resulting subset [Jaw12]. N-Random-Sampling can also be used to validate the semantic representation of a column with word embeddings: for this

purpose, sample sets containing randomly selected instances taken from the column are compared and should have a cosine similarity close to one.

**N-Most-Common Sampling (with distinct sampling):** Select the $n$ most frequent (distinct) instances to compute the common representation. This method will discard rare values and ensure that even frequent placeholders will only be included once in the representation. Due to the computation overhead, this method is only beneficial if there are certain instances that occur considerably more frequently than others.

After a representation has been found for the individual attributes, all attributes of the possible source and target table must now be compared to each other. Depending on the task, a user can tune different parameters for that—e.g., to choose whether tables should only be matched when they contain mostly the same types of data, or when one is more fine-grained than the other, too. Different versions of aggregated scores over whole tables can then be used to find full or partial matches.

Our experiments (see Section 14.5, Experiment 2) show that our approach works fine for an interactive approach, where the system pre-filters potential matches and the user then confirms the correct match from a short list of possible table pairs. In our experiments, filtered lists of an average length of $5$ entries will in about $75\,\%$ of the cases contain the correct correspondence. A qualitative analysis shows that the representation of columns containing abbreviations or artificial IDs often resemble each other. At this point, an additional syntactical comparison for exact matches could help to distinguish between them. A combination of schema- and instance-based matching strategies can be used to reduce the necessary amount of computation, since the runtime of instance-based matching will scale quadratically with the number of attributes and additionally depends on the number of instances.

The experiments show that the described procedure can be used to find correspondences that would not be recognized by syntactic matchers without, e.g., requiring domain-specific ontologies. With the table matching step, it is possible to severely restrict the set of attributes to be compared, thus to considerably reduce the calculation time for attribute matching or to allow more complex operations to be performed per comparison.

#### Step 2: Attribute Matching

In order to determine not only tables that might contain related content, but also exact correspondences between attributes of different tables, the candidates from the previous step must now be refined. Our approach can both be used to create 1:1 relationships or a list of possible attribute correspondences, which can then be used directly or presented for feedback. Table correspondences from table matching for which no attribute correspondence could be confirmed are automatically rejected.

We can re-use concepts from table matching, but need to at least adapt weights and thresholds. As before, both structural information (such as table and attribute titles, or comments) and the actual instances can be used for this purpose.

A simple approach is name-based attribute matching, which only considers the names of the individual attributes without including any additional information like neighboring elements or

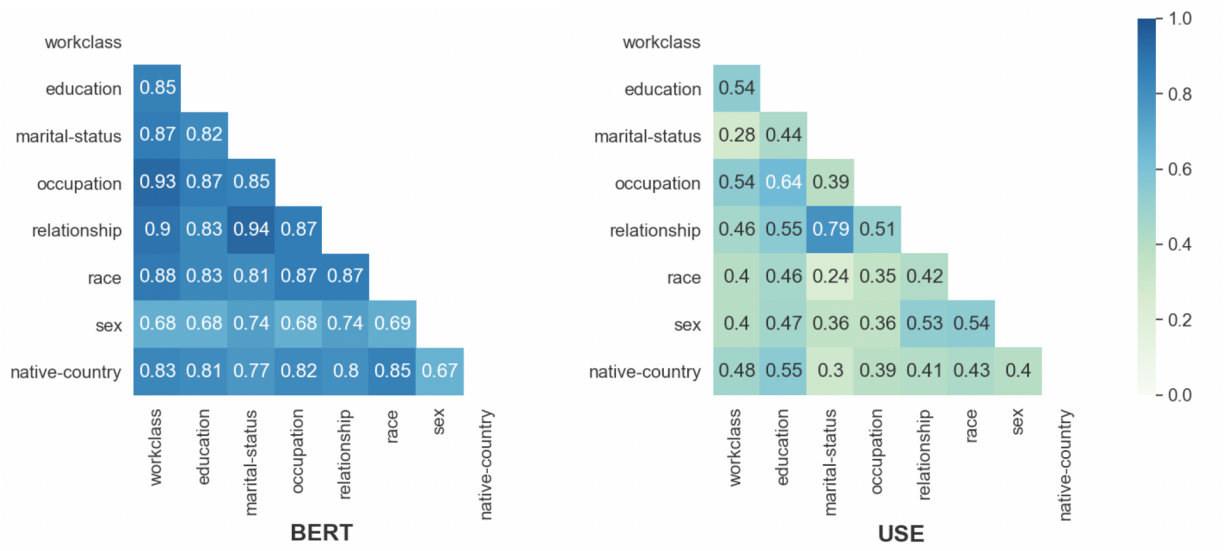



Figure 5.3.: Cosine similarities between the instance representations of the attributes of the *Adult* dataset for representations based on BERT and Google USE. For BERT, even dissimilar attributes have similarity values of at least 0.65, while for Google Use the range is much wider. Published in Hättasch et al. [Hät+20b].

data type information. The similarity of the attribute-matcher thus purely relies on the cosine-similarity of the embedding representation for two attribute names.

Moreover, analogous to the approaches introduced in the last section, the instances can be used for attribute matching, too. To do so, we again combine the representations of all or a sample of the entries of an attribute. We can show that our attribute embeddings are robust when sampling; i.e., the representations based upon different samples from the instances of an attribute are similar to the one for another representation of the same attribute based upon a different sample (see Section 14.6, Experiment 6). However, it is not sufficient that the representations are robust even with sampling, they must also differ as much as possible from the representations of the other attributes. Here, our experiments (see Section 14.6, Experiment 7) show that sampling is useful for spreading the range of values and thus ensuring that dissimilar attributes have a lower similarity value while the similarity value of related attributes remains high.

Moreover, the (dis-)similarity of the attribute embeddings depend on the pre-trained embedding method used. In Figure 5.3, one can see the similarity between the representations for different attributes of the *Adult* dataset. We use BERT embeddings on the one hand and representations based upon Google USE on the other. With BERT, even very different attributes have similarity metric values of at least $0.65$, while the range is significantly larger when using USE, making it much easier to automatically tune suitable thresholds.

Our experiments (see Section 14.6, Experiment 5) further demonstrate that it is reasonable to not perform the handling of multi-word expressions and domain-specific vocabulary through additional steps, but to use contextualized embeddings that can handle them inherently.

Finally, we compare our different strategies to state-of-the-art approaches provided by the COMA 3.0 framework [DR02]. Details on our settings, the baseline approaches and all numbers can

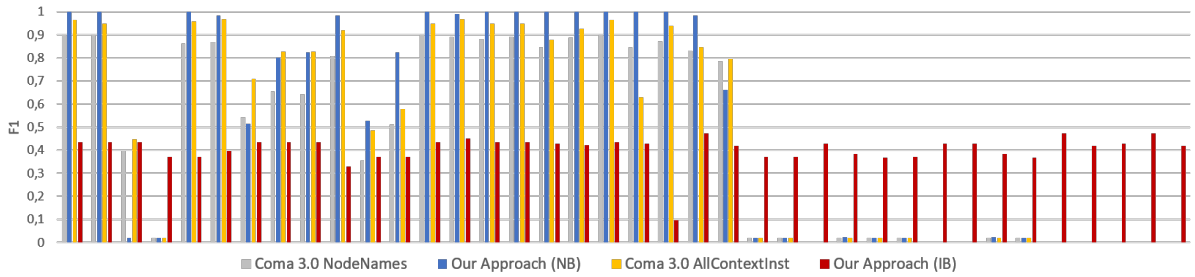



Figure 5.4.: F1 Scores for the name-based (NB) and instance-based (IB) variant of our approach to all 38 matching problems of the OAEI benchmark, compared matchers from the COMA framework. Values between 0 and 1, higher is better. Published in Hättasch et al. [Hät+20b].

be found in Section 14.6, Experiments 4 and 8. The resulting F1 scores for our two matching approaches and the baselines on $38$ matching problems from the *OAEI benchmark* can be found in Figure 5.4. We show the results of our approaches applied stand-alone, even though there are no instances to use for about two thirds of the attributes. For productive use, however, it would be advisable to combine this approach with a name-based approach and prepend a pre-selection step based upon the availability of instances.

It can be seen that our name-based approach almost always performs better for those problems where the name-based variant of COMA already performs well, but performs worse for problems where COMA already has difficulties. Our instance-based approach provides very high precision of $0.98$. The seemingly low recall of $0.26$ can be explained by the sparsity of instances. Most of the correspondences of these attributes can be found by the instance-based matcher, which can roughly maintain its quality-level even for the problems where the other three approaches have severe difficulties. Instance-based matching can therefore help to find correspondences when attribute names are not meaningful or semantically difficult to compare, for example due to domain-specific language. Overall, one can assume that a balanced combination of our two approaches would beat both the simpler and the complex (combined) baseline approach.

As a reference for the determination of the computation time for semantic representations with Google USE, we measured the runtimes for the *RT* dataset on a server with NVIDIA® TESLA® V100 graphics card with 16GB memory. The implementation of the instance-based and thus most complex matching took under three minutes to calculate all column vectors of the dataset (which has 10 columns with an average of $10,000$ instances).

### Summary

To summarize, we find that neural word embeddings can be utilized to propose a small set of possible candidates for schema matching, which is crucial for data integration. Word embeddings can be used to bridge the semantic gap for several matching variants. Our approach can be used instead of but in particular as a supplement to existing syntactic and semantic matchers. The use of models pre-trained for general tasks seems to be sufficient, as long as the database does not predominantly contain abbreviations and very specific terms. It is advisable to use contextualized embeddings.

Both structural data like schema information and comments as well as the textual data instances themselves can be used for matching, whereby the effectiveness of the individual approaches strongly depends on the schemata. In instance-based approaches, sampling can help to increase the distinction between similar and dissimilar attributes while reducing the number of instances to be considered. This also makes it possible to use the approach for databases with very large numbers of instances. Instance-based approaches work well with different types of entities. Weaknesses are found, for example, with attributes that all contain human names: the embeddings are good for finding other attributes with names, but a further subdivision (e.g., between actors and directors) is difficult.

## 5.2. Ad-hoc Information Extraction & Organization: WannaDB

Before, we showed how existing structured representations can be combined to match the users' needs. The right information at hand allows wise decisions, whether in research, engineering, finance, healthcare, or other fields.

One example would be a journalist, writing an article about the safety of electric cars. To prove that electric cars are not causing more fatal fires than those with a combustion engine [Dia23], they have to extract the mode of drive and accident details from numerous reports of car accidents.

Another example is research and development, where progress can only be made by taking previous findings into account. Academic publishing companies have realized that simply providing access to papers and books is not enough to meet the needs of researchers, engineers, and other professionals, which are often looking for certain details in a specific subfield, and that in a quick-to-use representation. As a result, research platforms like *SpringerMaterials*[7] are built, providing enhanced and interactive visualizations, tabular overviews, and domain-specific search functions backed by extensive and carefully curated databases. They offer users to directly access values, e.g., certain material properties, instead of having to gather them from different parts of, e.g., the more than 350 volumes of the *Landolt-Börnstein* book series that is one of the central data sources for this material sciences platform.

However, creating such a platform requires an enormous (manual) effort, and users from other domains can therefore not expect that a similar resource exists for their information need. The amount of knowledge humans accumulated grows with an unprecedented speed [Ful82; Sch13]. Researchers might have access to hundreds or thousands of relevant pages of text,[8] but it is not possible to process all of them manually. Instead, they need a tabular representation that is built from extracted or aggregated representations of the contents with a certain user-defined focus. Most existing data discovery techniques require technical background knowledge and are not easily accessible to domain experts from, e.g., finance or healthcare. Thus, when thousands of papers were published to get the COVID-19 pandemic under control [Har+20], large amounts of numbers had to be manually collected from patients files or status reports, to uncover trends and assess efficiencies. Automating data collection in a way that it is usable for medical researchers and professionals without a data or computer science (CS) background allows them to concentrate less on manual extraction work and more on treating people or drawing conclusions.

[7] `https://materials.springer.com`

[8] Thanks to the progress of open access, every second new publication is now provided openly, and the number is growing [Piw+18]

We therefore propose an approach that puts domain experts and knowledge workers in the driver's seat, acting as their own personalized data scientist. Our system allows them to interactively extract domain-specific information from text collections, organize it in tabular form, and find answers without requiring a trained data scientist. We leverage our knowledge about embeddings and the use of concrete instances for automatic alignment as described in Section 5.1 to achieve this further democratization level of exploration.

**Publications:** We published this work in multiple publications. The overall vision and general idea was published as "Benjamin Hättasch. 'WannaDB: Ad-hoc Structured Exploration of Text Collections Using Queries'. In: *Proceedings of the Second International Conference on Design of Experimental Search & Information REtrieval Systems, Padova, Italy, September 15-18, 2021*. Volume 2950. CEUR Workshop Proceedings. CEUR-WS.org, 2021" (see Chapter 15).
A first implementation was then presented and evaluated in "Benjamin Hättasch, Jan-Micha Bodensohn, and Carsten Binnig. 'ASET: Ad-hoc Structured Exploration of Text Collections'. en. In: *3rd International Workshop on Applied AI for Database Systems and Applications (AIDB21). In conjunction with the 47th International Conference on Very Large Data Bases, Copenhagen, Denmark, August 16 - 20, 2021*. Copenhagen, Denmark, 2021. arXiv: 2203.04663" (see Chapter 16). The paper proposed a two-phase approach for the core component to extract information and match it to the target attributes. In this paper, we propose a new system that allows users to perform structured explorations of text collections in an ad-hoc manner. The main idea is to use a new two-phase approach that first extracts a superset of information nuggets from the texts using existing extractors such as named entity recognizers and then matches the extractions to a structured table definition as requested by the user based on embeddings.
Later, we published a demo paper as "Benjamin Hättasch, Jan-Micha Bodensohn, and Carsten Binnig. 'Demonstrating ASET: Ad-hoc Structured Exploration of Text Collections'. In: *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. ACM, 2022" (see Chapter 17). This demo features the system with a graphical user interface that allows people without machine learning or programming expertise to explore text collections efficiently. For the demo paper, we created a completely reworked version of the algorithm, that we again improved for the full paper "Benjamin Hättasch, Jan-Micha Bodensohn, Liane Vogel, Matthias Urban, and Carsten Binnig. 'WannaDB: Ad-hoc SQL Queries over Text Collections'. In: *Datenbanksysteme für Business, Technologie und Web (BTW 2023), 20. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 06.-10, März 2023, Dresden, Germany, Proceedings*. Volume P-331. LNI. Gesellschaft für Informatik e.V., 2023" (see Chapter 18). This features the full application cycle including target structure deduction, interactive table extraction, interactive grouping, and result computation. It features an extensive evaluation and not only earned the reproducibility badge but was also awarded as best long paper of BTW'23.
Our latest progress for the query execution procedure and the user interaction is contained in a paper currently under submission as "Benjamin Hättasch, Liane Vogel, Gard Jenset, Jan-Micha Bodensohn, Chandrima Roy, and Carsten Binnig. 'WannaDB in Action: Deploying Ad-hoc SQL-over-Text Exploration in an Industrial Scenario'. In: *Under submission* (2023)" (see Chapter 19). In this paper, we showcase the potential real-world deployment of our tool in an industrial scenario and conduct a multipart user study to prove its usefulness for real world applications.

**Contributions of the author:** The research on *WannaDB* was carried out over the course of more than three years, with me being the project lead. I was therefore responsible for selecting and

developing approaches to try, selecting research directions, and evaluating them. The research on this project was partially funded by the Softwarecampus program. That allowed me to hire three people supporting me with the research. Substantial parts of the implementation were done by Jan-Micha Bodensohn, who was working first as a bachelor thesis student and then as a student helper, both under my supervision, on the project. Liane Vogel worked as Ph.D. student part-time on the project and was supervised with regard to it by me, too. She contributed to the literature research and carried out parts of the evaluation. Jonas July joined the team for a few months as an additional student helper and contributed to the implementation of the demo system. As part of the program, I also discussed and evaluated potential real-world application scenarios and use-cases with the industry partner *Springer Nature*, in particular with Gard Jenset, my contact there. As the project lead, I was responsible for selecting the publication strategy, planning the contents of each publication, and creating major parts of the manuscripts. The co-authors Jan-Micha Bodensohn, Liane Vogel, Gard Jenset, Matthias Urban, Chandrima Roy, and Carsten Binnig contributed invaluable feedback. All authors agree with the use of the publication for this dissertation.

### 5.2.1. Motivation, Challenges & Idea

One example of an extraction task that can be tackled with our system is financial news-gathering, where analysts must quickly obtain key figures like revenue, earnings per share, or mergers and acquisitions from company reports and press releases. Unfortunately, this task is cumbersome and error-prone when done by hand. Furthermore, off-the-shelf extraction systems often cannot extract all information the user is interested in. As domain experts and knowledge workers typically do not have the technical skills to craft custom extraction pipelines, they might experience a *context gap*, i.e., they are unable to bridge between their specific terminology and the labels used by existing extraction approaches. Moreover, knowledge workers are often not allowed to send sensitive data to third-party APIs, so one cannot always expect to have a domain-specific pre-trained language model at hand. Learning a dedicated extraction model for such a task is often infeasible, since domain practitioners often lack both the required training data and the ability to train or fine-tune a model. Finally, a learned model is in most cases not interpretable by humans, leading to a lack of trust in the results [Sch+22].

These problems are even more significant for the open-ended exploration of text collections, where the relevant attributes are not known upfront. Users need a data-first approach that allows them to quickly get insights without long feedback loops. Again, being able to quickly organize the data in tabular form can be of great help. Such a table may serve as the basis for further exploration and analysis, for example by revealing subsequent avenues of investigation, by providing initial answers through the computation of aggregates such as sums or averages, or by further filtering the document collection based on particular attribute values.

With *WannaDB*, we present such a tool that enables users to derive tabular representations from text collections and even run SQL-like queries on it. *WannaDB* extracts and organizes information by incorporating user feedback. Instead of defining manual extraction rules (like, e.g., in [GB19]), users only need to confirm or rectify a small amount of guessed extractions.

Through user interaction, *WannaDB* gives users the opportunity to steer the extraction process and does not require large amounts of training data to adjust to new domains. Compared to trained black-box extraction models, *WannaDB* transparently shows the user what is extracted from the

texts and gives them the opportunity to fix incorrect extractions right away. Approaches like this are supported by a recent study [Sch+22], that emphasizes the importance of keeping the human in the loop to both increase efficiency and trust in tools and results. Moreover, this will get even more relevant in the future as regulations such as the *European General Data Protection Regulation* or the upcoming *AI Act* include a "right to explain".

The granularity of information that can be possibly extracted by *WannaDB* goes far beyond the one of classical named entity recognition and, in particular, incorporates the context in the target definition (e.g., finding the pharmaceutical company conducting a clinical trial as described in the current document vs. finding companies or even all kinds of organizations in that text). *WannaDB* even manages to do so for numeric values with different semantics, e.g., finding both the number of people vaccinated once and twice in daily status reports on the COVID-19 pandemic.

One could now argue that the task of information extraction from text could easily be addressed using large language models such as GPT-3 [Bro+20] or LLaMA [Tou+23]. However, their high resource requirements and long runtimes currently inhibit their use in exploratory settings, as we demonstrate even for smaller models such as BART [Lew+20]. In contrast, *WannaDB* requires only a one-time pre-processing per text collection on a single GPU, after which the text collection can be explored at minimal cost on typical consumer hardware (even on CPU-only machines). In addition, language models are often accessible only through an API and employ pay-per-query business models, where users are reluctant to openly explore the data as they have to pay for every single query. By contrast, *WannaDB* is released as an open-source tool that domain experts can apply directly to their own data.

Finally, our approach combines the advantages of language models or embeddings with the strict calculation possibilities and deterministic behavior of the SQL data query language. It can produce tables stating information that is not explicitly mentioned in the documents and hence not discoverable by pure extraction or search approaches, and can perform numerical reasoning on the data without the need to rely on the limited mathematical abilities of a language model [Hen+21].

### 5.2.2. Design Considerations

First, we aim to provide insights into the relevant design considerations we made. Text collections from various domains can have substantial differences, but the task of extracting attributes from the texts and bringing the results into a structured form always follows the same steps. Whether a user needs to extract data from COVID-19 status updates or patient files, different text collections have certain things in common:

1. The available data is usually focused around a topic.
2. That data mostly follows a certain structure (similar chapters/sections, level of detail, length).
3. The language used in the text collection is usually quite homogeneous, but might be strongly specialized for a certain topic.
4. Often, a clear distinction which part of the collection covers which event or entity is possible (e.g., each incident report covers one incident, each patient file reports on one patient, each paragraph in an episode guide describes one episode of a TV series).

5. In many cases, questions to a text collection are of exploratory nature, i.e., they are required to find out what should be investigated further (e.g.: Are there certain noticeable incidents? Or should one continue the exploration with a new hypothesis?).

These considerations result in several challenges, but also chances to tackle the problem of information extraction from text collections:

1. The homogeneity of language and structure inside a document collection enables to build systems that automatically extract and organize the relevant information. Yet, the heterogeneity across document collections makes it very hard to construct a *one-fits-all* approach.
2. Since the data is often already partitioned or can simply be split into documents covering a single topic at once, we can assume that one document corresponds to one row in the target table (e.g., covering properties of different airline incidents or the situation on different days).
3. The exploratory nature of the queries requires a quick and cheap answer, but also allows this answer to be approximate, as this will still be sufficient to quickly decide whether an avenue of investigation is promising or not.

We therefore decided to address the task with an approximating approach that tries to quickly generalize simple domain-specific information. Thereby, we massively lower the table building costs in trade for exactness of the created table that the user probably does not need for exploratory tasks. Furthermore, we exploit the semantic partitioning of the text collections into documents and create one table row per document. The extracted table can be seen as the materialized result of a join, we will extract a wide table (e.g., containing information about an incident itself but also the airline and airport involved) which the user might split if needed instead of requiring the computer to make assumptions about the schema. Finally, we concentrate on the current need of the user, thus our approach works lazily—instead of creating a complex knowledge base upfront, tables are created and filled only when the user needs them for the first time.

### 5.2.3. Exemplary Usage

In this exemplary usage scenario, we aim to show how *WannaDB* can be used to satisfy an information need based on a text collection. Imagine, e.g., a data journalist who just obtained a large collection of airline incident reports and is now looking for noticeable events, like a high rate of incidents for a certain carrier or airport. They use *WannaDB* for that purpose. The data journalist starts by loading the collection of text files into *WannaDB* and triggers the pre-processing of the files, a process that needs to be done only a single time for each text collection.

Next, the data journalist enters a SQL-like query as a starting point for their exploration (e.g., `SELECT airline, airport, COUNT(*) GROUP BY airline, airport`). As there is no pre-existing table yet, the FROM-part of a typical SQL query can be omitted, simplifying the query syntax. After entering the query, *WannaDB* presents a list of possible matches for each required attribute (e.g., airline) found in texts of the collection, as shown in Figure 18.2. Not all found matches will be correct right away, therefore *WannaDB* relies on some user input to adjust the results. The data journalist confirms a few of the correctly found matches, corrects wrong matches by choosing the relevant extraction or marks if the required attribute does not occur in a given text (see Figure 18.2). Meanwhile, *WannaDB* continuously updates the list of all guessed matches during this interactive phase, leveraging the feedback. The user interface allows to quickly

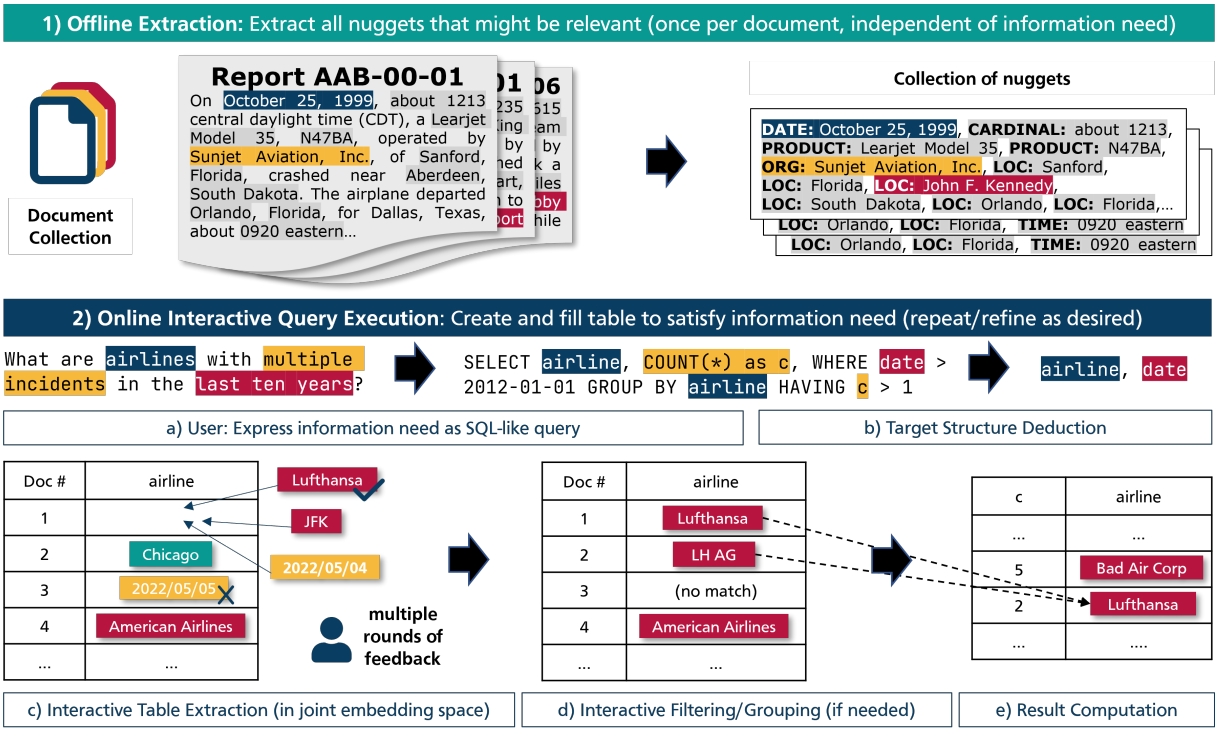



Figure 5.5.: Architecture & exemplary usage: The offline extraction phase obtains information nuggets from the documents. The online phase then infers the required structure from a query, matches between the extracted information nuggets and the user's schema, performs the grouping, and executes the query. Published in Hättasch et al. [Hät+23a].

identify entries that stand out and get an impression of the quality already achieved. Once the data journalist is satisfied with the quality of the matches, they continue with the next attribute of their query.

After all attributes are processed, *WannaDB* will execute the query on the resulting table. If the query contains grouping operations, the data journalist might be asked again for some interactive feedback (e.g., to confirm that *Lufthansa* and *LH* refer to the same airline, but *LHS* does not). *WannaDB* will again try to transfer this feedback to other rows. In the end, the data journalist will receive an answer to their query and can export the resulting table to a spreadsheet, a SQLite table, or a Pandas Dataframe for further investigation. If they have further queries to submit to *WannaDB*, the interactive matching process only needs to be repeated for new attributes, as *WannaDB* leverages existing results from previous queries.

### 5.2.4. Our Multi-Stage Approach

Our system tackles the problem using a multi-stage-approach with two stages: an offline stage to extract information nuggets (i.e. short information-bearing text snippets), followed by the interactive stage to answer the query by table extraction and if required interactive filtering or grouping. The overall workflow is visualized in Figure 5.5.

### 5.2.5. Stage 1: Offline Extraction

In the first stage we employ off-the-shelf information extractors to extract a superset of potentially relevant information nuggets (e.g., named entities) from the given text collection. This step is independent of user queries and can thus be executed offline to prepare the text collection for ad-hoc exploration by the user. The extractors process the collection document-by-document to generate the corresponding extractions. Clearly, a limiting factor of *WannaDB* is which kinds of information nuggets can be extracted in the extraction stage, since only this information can be used for the subsequent matching stage. As a default, we use named entity recognizers from *Stanza* [Qi+20] and *spaCy* [Hon+20]. In general, *WannaDB* can be used with any extractor that produces label-mention pairs; i.e., a textual mention of an information nugget in the text (e.g., *American Airlines*) together with a natural language descriptor representing its semantic type called label (e.g., *Company*). Moreover, additional information about the extraction (e.g., its position in the document and the surrounding sentence) is also stored and used for computing the embeddings, as we describe below.

After extraction, the information nuggets are pre-processed to derive their actual data values (i.e., a canonical representation, e.g., for timestamps) from their mentions. For this we also rely on state-of-the-art systems for normalization [Man+14]. The nuggets are then represented based on the following signals: (1) *label* – the entity type determined by the information extractor (e.g., *Company*),[9] (2) *mention* – the textual representation of the entity in the text (e.g., *Lufthansa*), (3) *context* – the sentence in which the mention appears, (4) *position* – the position of the mention in the document. Each information nugget representation comprises embeddings for the individual signals (1-4). We compute semantic representations for the natural language signals using FastText [Mik+18] (1), Sentence-BERT [RG19a] (2) and BERT [Dev+19] (3), and normalize the position by dividing it by the document length.

### 5.2.6. Stage 2: Interactive Table Filling

To produce the results for the users, the core of our system is filling a table with a structure explicitly or implicitly requested by the user. This requires automatically selecting a (the best) matching nugget from a document for a given attribute—or deciding to actively leave a cell empty if the document does not contain the requested information. To bridge between the information contained in the embeddings and user/domain specific terminology that is not reflected in generally available language models, we use user interaction/feedback that is then generalized. Our experiments show that the selection of which elements to give feedback to can drastically influence the number of interactions needed to reach a certain correctness of the table filling and thus the overall quality that can be reached with a reasonable amount of interactions. Hence, a good table filling algorithm should carefully steer the feedback process.

We therefore developed multiple versions of such an algorithm. Their common idea is exploring the embedding space to find the areas that probably contain the correct nuggets for a given attribute, and then using this knowledge to fill the cells for the remaining documents that the user did not give feedback to.

[9] We map the named entity recognizers' labels like *ORG* to suitable natural language expressions according to the descriptions in their specification.

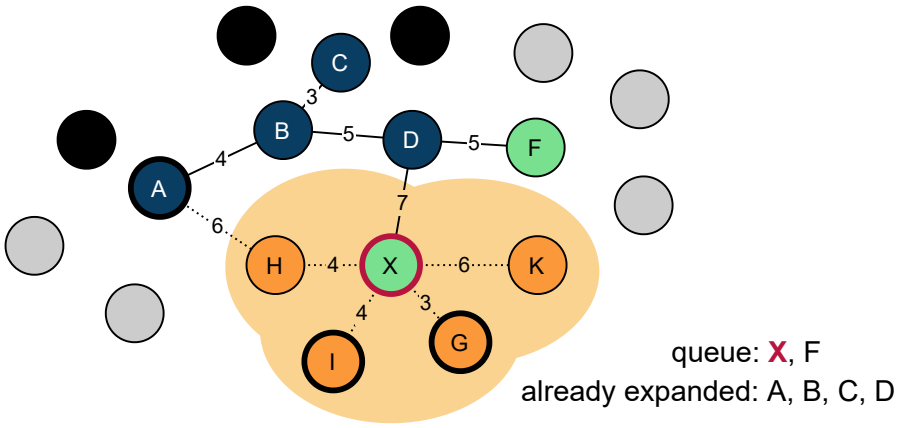



Figure 5.6.: Sketch of the tree-based (explore-away) strategy (executed per attribute). Each node represents an embedded information nugget to be included in the search tree. Confirmed matches are marked in ■ blue, the next candidates in ■ green. Rejected nuggets are marked in ■ black, unexplored ones in ■ gray. Node X is selected for expansion, the nodes closest to X are marked in ■ orange. The candidates selected by our explore-away strategy for user-feedback are G and I.

#### V1: Tree-based explore away strategy

Our first version therefore used a tree-based strategy, where the tree represents confirmed matching nuggets (instead of a set as often used by kNN-based approaches). Different from kNN-searches, subspace clustering, or other techniques tackling similar problems, using a tree-based representation (where each node represents a nugget from the input texts) allows us to implement a new *explore-away* strategy that can grow the covered embedding space for the group of related information nuggets with every confirmed match. Our tree-based exploration strategy works in three steps:

**1. Find a root node:** First, the exploration strategy finds an initial matching node to serve as the root of the tree. This is done by sampling extractions based on their distance to the initial attribute embedding (based only on the attribute name). We start with low distances that result in conservative samples close to the initial attribute embedding and gradually raise the sampling temperature to include samples from farther away if the close-by samples do not yield any matching extractions to select as root.

**2. Explore-away Expansion:** As a second step, we now explore the embedding space by expanding the search tree using our explore-away strategy in the embedding space. We explain the expansion step based on the example in Figure 5.6 where node $X$ is to be expanded. To expand that node $X$, we determine its potential successors $\texttt{succ}(X)$ based on the following two constraints: (1) The extractions in $\texttt{succ}(X)$ must be closer to $X$ than to any other already expanded extraction (e.g., nodes $G$, $H$, $I$, and $K$ qualify in our example). (2) The extractions in $\texttt{succ}(X)$ must be further away from the rest of the tree than the node we expand (e.g., $H$ is closer to $A$ than $X$ is to its parent (and hence closest node) $D$ and therefore not a candidate; however, nodes $G$, $I$, and $K$ remain as candidates).

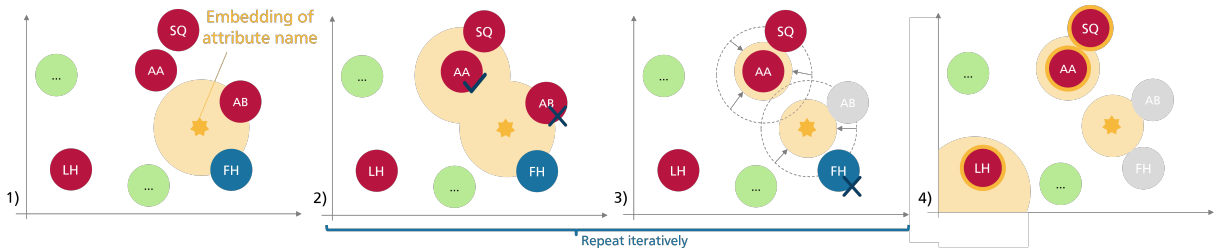



Figure 5.7.: Schematic visualization of the new distance-based table filling algorithm.
1) WannaDB presents initial guesses (distance to attribute name embeddings)
2) The user gives feedback for one entry (row)
3) WannaDB updates its guesses (distances to confirmed elements) and (in later versions) tunes thresholds and adds new nuggets
4) Once the user is satisfied with the quality, remaining rows are filled with best guesses

Afterwards, the search strategy selects the $k$ nuggets[10] in $\mathsf{succ}(X)$ that are closest to $X$ (e.g., $G$ and $I$ in our example) to gather user feedback. A user can then confirm whether the proposed nuggets actually match the attribute; matching nuggets are added to a queue of nuggets to be expanded in the next iterations. In case the queue is empty, the explore-away strategy returns to step 1 to start with an additional root node, or it terminates if a user-defined threshold of confirmed matches is reached. Overall, this procedure thus identifies groups of information nuggets (represented as trees) that match to a certain user-requested attribute.

**3. Static Matching:** Once a user-defined threshold of confirmed matches is reached for every user attribute, the system stops collecting feedback and continues with a static matching procedure: the system leverages the distances between the embeddings of extracted information nuggets and the different groups of embeddings identified in step 2 to populate the remaining cells without asking the user for feedback.

More details can be found in Chapter 16. This table filling strategy already provided good results on many attributes (see Section 5.2.7, but left rooms for improvements for others. Probably even more important was, however, the experience of the user when using the system. By requesting a yes/no feedback from them, there was no possibility to actively point the system to the correct nugget for a cell, even if the user already saw it in the text snippet they had to read to give feedback. We therefore completely reworked our algorithm for the second publication.

### V2: Distance-based Version with Rich Feedback

For the second version of our approach, we introduced new feedback actions: Instead of giving yes/no-feedback on a single attribute, the user is presented with a document with the best guess for the matching nugget preselected. They can now either confirm that nugget, select another one from the highlighted parts of the text, or state that there is no matching nugget in the text at all.

[10]This number determines the degree of the search trees. We experimented with different degrees and found that 2 results in the best performance.

As a second major change, we move away from completely automatically selecting which elements go give feedback to. Instead, we leverage the human ability to quickly identify those elements in a bunch that stand out, by presenting a list of, e.g., 10 probable matches together with the sentence around them and letting the user decide which of these to confirm or correct. This list can be filled by different strategies, as we will discuss below.

The new version populates the user table in an attribute-by-attribute fashion, too. In the following, we describe the table filling procedure for an attribute called *airline*, a schematic visualization of the steps can be found in Figure 5.7, in Figure 5.9 we show a pseudocode representation of our algorithm (including the extensions we will describe in the next sections).

For each information nugget, the system caches a distance that represents the (un-)certainty with which it believes that the information nugget matches the attribute. At first, this distance is initialized as the cosine distance between the nugget's label embedding (e.g., *Organization*) and the embedding of the attribute name *airline* (step 1 in the figure). Later on, this distance will be updated with the distance to the closest confirmed matching nugget, allowing the system to capitalize on more signals like the textual mentions (e.g., *American Airlines*) of other matching information nuggets.

For each document with no confirmed match, the system considers the information nugget with the lowest cached distance as the currently guessed match. The system presents a list of the documents with the most uncertain current guesses to the user for feedback. The user can then provide feedback for any of these guesses (step 2). They may either confirm the match, select another information nugget from the document as the match, or state that the document does not contain a matching information nugget. In case their feedback results in a confirmed match, this matching information nugget is used to update the distances on all other remaining information nuggets (step 3). Since the distance between each nugget and its previously closest confirmed match is cached, only one new distance (the one between it and the recent match) needs to be calculated for this update. Therefore, our new version has a complexity of $\mathcal{O}(n)$,[11] leading to fast updates after the interaction even when scaling up the number and length of input documents (see Section 5.2.7). Afterwards, the document is removed from the list of remaining documents, and the list of guesses presented to the user for feedback is updated.

Finally, the user may decide (based on the list of most uncertain guesses that is presented to them) to terminate the interactive table filling procedure and continue with the next attribute, in which case the remaining documents' cells will be populated with their currently guessed matches (step 4). Thus, for each document without a manually confirmed nugget, *WannaDB* fills the cell with the information nugget with the lowest cached distance. Furthermore, *WannaDB* uses a distance *threshold* for each attribute to decide when a cell should be left empty instead.

With the new version of the approach, we were able to improve the quality that could be reached with the same amount of interactions (see Section 5.2.7).

An intuition that our system indeed explores the vector space and leverages closeness can be found in Figure 5.8. It shows a low-dimensional projection of the guesses of the system after different numbers of feedback iterations compared to the gold standard shown in the same way. It can be seen that the selection quickly converges towards the correct values.

[11] With regard to the number of nuggets.

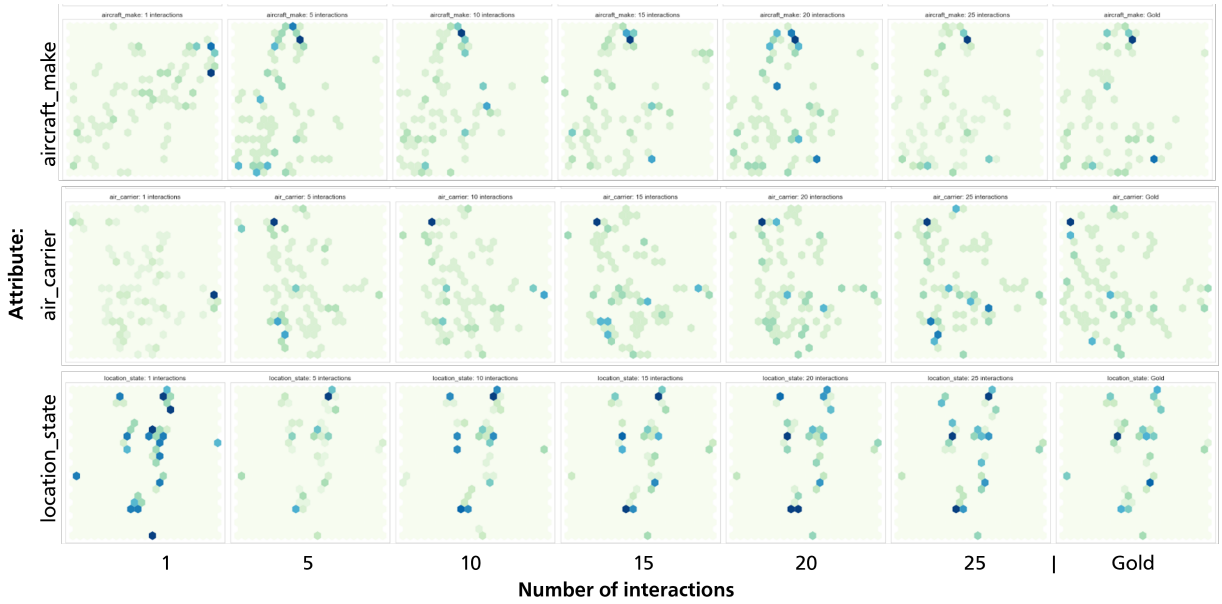



Figure 5.8.: Visualization of the explored vector space for three attributes from the aviation dataset. The important dimensions for each attribute are highlighted using UMAP and plotted as a hexbin visualization, showing the guesses of *WannaDB* after 1, 5, 10, 15, 20, and 25 interactions compared to the gold selection in the last column. Adapted from Hättasch et al. [Hät+23b].

### V3: Advanced Selection Strategies & Automatic Threshold Tuning

For this version of the algorithm, we evaluated multiple strategies to fill the list that *WannaDB* asks for feedback for. Moreover, we introduced automatic tuning for the distance threshold that before was a hyperparameter.

In the interactive table filling phase, *WannaDB* presents a ranked list of documents with their currently guessed matches to the user for feedback (see Figure 18.2) and will continuously update the list after every given feedback. This allows the user to quickly identify (incorrect) entries that stand out and to get an impression of the quality already achieved. After evaluating different strategies, including a random one and one sampling stratified from the distribution of distances, we settled on the following strategy: The ranked list is centered around the maximum distance that a nugget may have to a confirmed one to still be considered a valid value for a table (called *threshold* in the following) and thus presumably shows both correct guesses with a low certainty, and incorrect guesses, where *WannaDB* would profit most from feedback. Each feedback is processed as described in the previous section, additionally, the extended version adjusts the threshold after each interaction accordingly.

To summarize, *WannaDB* uses a threshold for two purposes: (a) to decide when it is better to leave a cell empty than to use a very unlikely guess (mostly because the desired value is not mentioned in the document) and (b) to select guesses to present to the user where feedback will have as much effect as possible. This threshold is automatically tuned during the runtime of *WannaDB* to fit the data at hand. Given the approximate query setting *WannaDB* is built for, we decided to use

```
for attribute in query.attributes: # Process each attribute separately
  for nugget in all_nuggets:
    nugget.distance = compute_distance(attribute, nugget) # Compute initial
    ↪ distances

  while interactive_feedback_phase:  # Interactively get user feedback
    ranked_list = make_ranked_list(threshold, documents)
    feedback = get_user_feedback(ranked_list)
    match feedback:
      # Positive feedback (confirmation, manual correction, or custom selection):
      case ConfirmNugget(document, selection):
        confirmed_nugget = get_nugget_for_selection(selection)
       if confirmed_nugget is None: # The selection does not correspond to a nugget?
          confirmed_nugget = create_and_add_nugget(selection)
          find_similar_nuggets_in_other_documents(selection, document)
        # Mark this particular cell as manual confirmed...
        set_match(document, confirmed_nugget)
        # ... and update distances for all nuggets based on user feedback
        for nugget in all_nuggets:
          new_distance = compute_distance(nugget, confirmed_nugget)
          nugget.distance = min(new_distance, nugget.distance)
      # Negative feedback:
      case NoMatchInDocument(document):
        # Direct effect only on the given document...
        leave_empty(document)
    update_guessed_matches(documents)
    adjust_threshold(feedback)  # ... but both feedback types can have effects
    ↪ indirectly through threshold adjustment on other document's rows, too

  for document in documents:  # Only consider values up to a given maximum distance
    if current_guess(document).distance < threshold:
      set_match(document, current_guess(document)) # compute final result table
    else:
      leave_empty(document)

def adjust_threshold(feedback):  # Feedback can be further exploited in certain
↪ cases
  match feedback:
    case ConfirmNugget(document, confirmed_nugget):
      if confirmed_nugget.distance > threshold:
        increase_threshold(confirmed_nugget)
    case NoMatchInDocument(document):
      if current_guess(document).distance < threshold:
        decrease_threshold(document)

def decrease_threshold(document):  # Consider fewer matches as valid (especially
↪ those above last marking as incorrect that are currently accepted nevertheless)
  nuggets = ranked_list.between(threshold, document)
  min_dist = min(n.distance for n in nuggets)
  threshold = min(min_dist, threshold)

def increase_threshold(confirmed_nugget):  # Consider more matches as valid
↪ (especially those below last confirmation that are currently discarded because
↪ of the threshold)
  nuggets = ranked_list.between(confirmed_nugget, threshold)
  max_dist = max(n.distance for n in nuggets)
  threshold = max(max_dist, threshold)
```

Figure 5.9.: Pseudocode representation of our interactive algorithm for table extraction, including threshold adjustment. Adapted from [Hät+23a], in contrast to the published version the interactive adding of custom selections was integrated.

a common threshold for all regions forming in the embedding space instead of individually tuning it, to keep the number of interaction cycles low.

The adjustment of the threshold is shown in Figure 5.9 (line 34-51). The general idea is to incorporate the additional knowledge gained from the user confirming a nugget even though it was above the threshold or correcting an entry below the threshold. This feedback action will only affect a specific nugget directly, but other similarly well fitting nuggets from other documents might still be wrongly accepted or discarded because of the threshold, which is therefore carefully adapted after feedback actions: If the user confirms a nugget from the ranked list that is above the threshold, all nuggets between the threshold and this nugget should be considered as a good guess. In the case that any of the nuggets is still above the threshold after the calculation of the new distances, the threshold is adapted accordingly. In contrast, if the user states that for a nugget with a distance below the threshold there is no match in the document, the threshold is decreased to also exclude other matches that are in the list above the nugget if necessary. The threshold is only adapted in these two cases, where implicit hints about the quality assessment by the user can be incorporated.

#### V4: Interactive Adding of Nuggets

While the two-stage approach makes it possible to run our table filling strategy by providing a defined and restricted list of nuggets to calculate distances in the vector space between, it also poses a severe limitation: only those elements extracted in the first place can be used to fill the table, even when the user manually inspects a source document and sees the correct entry, they can not select the “correct” value if that one was not extracted in the first stage. Since that will both negatively influence the quality and be unsatisfactory for the user, we started to implement countermeasures in later versions of the project. At first, we allowed selecting arbitrary text spans when manually correcting a potential match. Next, we treated these spans like automatically generated extractions and added them to the vector space such that other close nuggets to this manually confirmed one could be found. Additionally, *WannaDB* searches for exact and similar occurrences of that text span in other source documents and treats them the same way, to go further beyond the manually corrected document. This version of the interactive matching is covered in Chapter 19. A pseudocode representation of our final algorithm version can be found in Figure 5.9. As part of our current research, we investigate how the user feedback can be further generalized to find even more potentially relevant, but previously unrecognized, nuggets during the matching step based on this kind of feedback.

### 5.2.7. Key Findings

In the following, we present the key findings from our work on *WannaDB*. As part of our evaluations, we conducted expert interviews, which underline the use of tools to extract and organize information without coding in exploratory or prototypic scenarios. We also did a small user study, where the users liked the ability to quickly try many queries, and favored that our system adapts to their use of terminology and not vice versa. Our experiments show that our system works both substantially better and faster than a learned few-shot system that received the same amount of domain information from the user. We can show how the interaction model we introduced directly influences the quality. Finally, we confirm that real users can achieve a similar result quality as

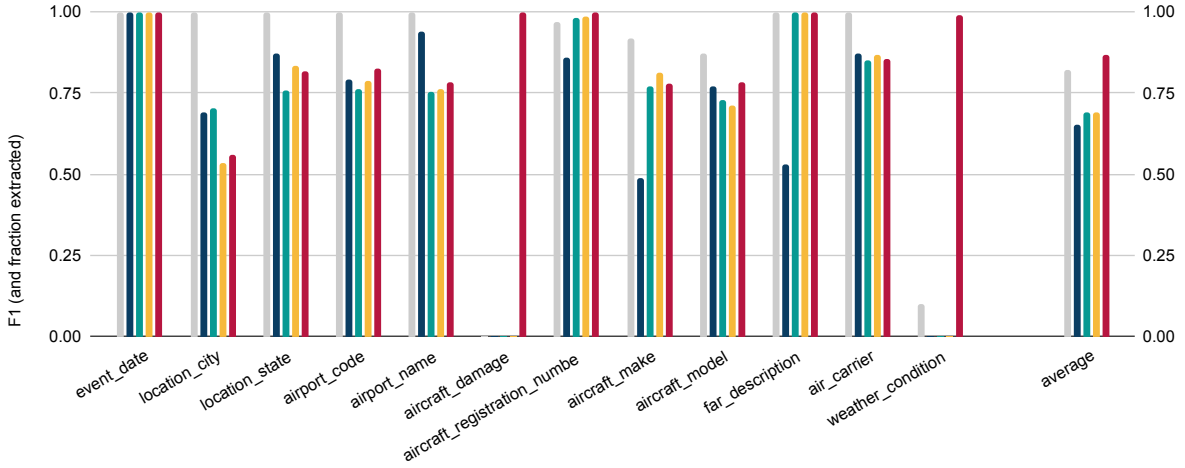



Figure 5.10.: F1 scores for different versions of our table filling algorithm for different attributes and the average of the aviation dataset. ■ tree based version, ■ first ranking based version, ■ with improved selection strategies and ■ with interactive adding of extractions at runtime. Additionally, the ■ fraction of required nuggets that are extracted by our default extraction stage are shown.
It can be seen that our algorithm improves over time. In particular, all versions achieve decent quality on all attributes that are mostly extracted, but only the ■ latest version can reach a high F1 score on all attributes, while the others fail on those where the extraction stage cannot sufficiently extract suitable nuggets in the first place. Values between 0 and 1, higher is better.

predicted in our simulations. The full evaluation of the end-to-end system as well as individual aspects and components can be found in the Sections 16.4, 18.6 and 19.5.

**Datasets:** For the evaluation of our system, we mainly used three datasets: *Aviation*, a text collection based on 100 aviation accident reports, *COVID-19*, again 100 documents describing the situation of the Covid-19 pandemic in Germany and focusing on numeric attributes with a lot of cases where the system should deliberately keep the resulting cell empty, and *T-REx*, an adapted version of the T-REx dataset [ElS+18] which is based on Wikipedia articles with aligned Wikidata triples from which we extracted three subsets (about Nobel laureates, countries, and skyscrapers). More details on the datasets can be found in Chapter 18. For each of these datasets, we either manually annotated the correct values for each attribute and document (Aviation, COVID-19) or used the existing alignment (T-REx).

### Evolution of the System Quality

The core contribution of our system is the interactive table matching algorithm. As described before, we continuously refined this component throughout our research. In Figure 5.10, we present the results of running those different versions on the same data and with the same number (25) of interactions per attribute. We use the *Aviation* dataset and the same default extraction stage for all

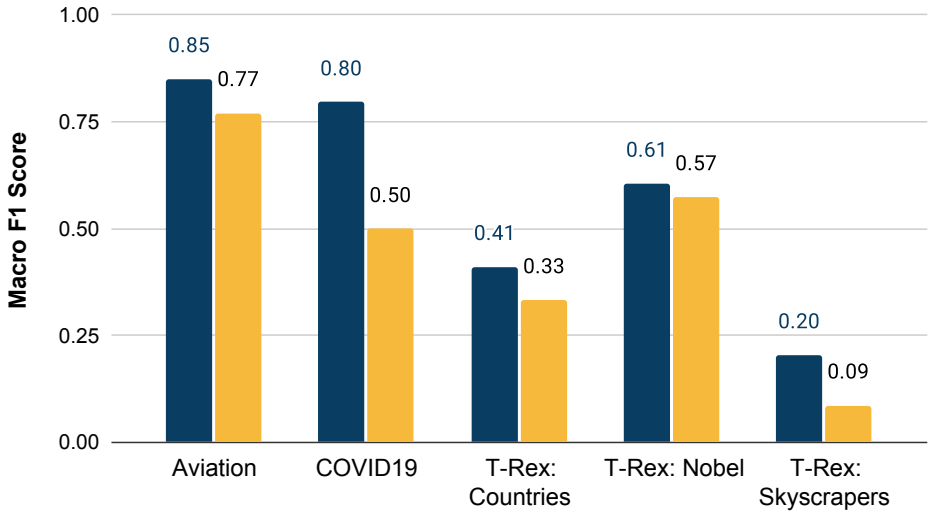



Figure 5.11.: Table filling results. It can be seen that ■ *WannaDB* outperforms the ■ few-shot BART model (fine-tuned on 20 datapoints) on every dataset. Published in Hättasch et al. [Hät+23b].

experiments. To visualize how the different versions react to the highly attribute-specific extraction stage quality, we visualize the fraction of the required nuggets that were successfully extracted.

It can be seen that even the first version (■) already provided high accuracy and recall (measured by the combining F1 score) for most attributes, on average $0.65$. However, for some attributes the table is filled with a much lower quality than for others or not at all (e.g., for weather conditions). One reason can be that the currently employed information extractors are not able to extract the necessary information nuggets from the text (■). In particular, *aircraft_damage* and *weather_condition* are examples where not only a large heterogeneity of mentions can be found, but also very domain-specific terminology is used.

With the distance-based approaches, we focused on providing lower runtimes and a better user experience. That caused slight decreases in the quality for single attributes. However, on average, the second and third version (■■) reach an even slightly better quality than the old version (F1 scores of around $0.69$).

To avoid having to rely on domain-specific extractors, we especially concentrated on dealing with missing extractions in the last version of our algorithm (■) and integrated a new interaction paradigm to improve. It can be seen that with the final version, those attributes are suddenly filled with a much higher quality. This final improvement boosts the average F1 score to $0.87$, which is able to outperform large trained models, as we will show below.

### End-to-End Quality

To measure the end-to-end performance of *WannaDB*, we simulate a fixed amount of user feedback (20 interactions per attribute) for a list of target table columns on each text collection and measure the resulting performance (F1 score, measuring correct filling or leaving cells empty) for each of

these attributes. We compare these scores to a few-shot training/refining approach that received the same amount of input/feedback, i.e., a BART [Lew+20] sequence-to-sequence model fine-tuned on 20 labeled examples, simulating the scenario of replacing the internal workings of *WannaDB* by a learned model. The details are explained in Section 19.5.1.

The results of our system in comparison to the few-shot baseline are shown in Figure 5.11. It can be seen that *WannaDB* consistently outperforms the few-shot baseline approach on all datasets. The highest performance difference is notable on the COVID-19 dataset, where *WannaDB* reaches an F1 score of $80\,\%$, whereas the BART baseline models only reach $50\,\%$. We suspect that this is partly due to the limited capabilities of language models such as BART when working with numerical values [Hen+21], as there are many numerical attributes contained in the COVID-19 dataset, such as the incidence or the number of patients in intensive care. Additionally, we noticed that the BART baseline models have difficulties when many of the texts do not mention the required values and thus many cells need to be left empty. Meanwhile, *WannaDB* shows consistently good performance extracting numerical attributes, as well as having fewer problems when dealing with missing data.

#### Effects & Costs of Interaction

A central measure to reach high quality results with low overhead (i.e., without hard or costly to acquire domain-specific resources), is the use of interaction. We therefore quickly show how interaction influences the correctness of the table filling on two of our datasets here. More details can be found in Sections 18.6 and 19.5.

First, we simulate the interactive matching process with different interaction limits (i.e., the number of interactions per extracted query attribute). To show the generalization abilities based on "simple" feedback, we disabled the adding of custom text spans for this experiment. We present the results on the *Aviation* dataset here, more details and the results for the other datasets can be found in Section 18.6.3. As we can see in Figure 5.12, for some attributes, *WannaDB* achieves very high F1 scores with only one interaction with the user (e.g., for *event date* or *aircraft registration number*). These are attributes where the entity type of the extracted information nugget is very similar to the attribute name or the pattern of the extracted information nugget is rather unique. For example, the extraction has the named entity tag *DATE* which is similar to *event date*. For other attributes though, the performance of *WannaDB* strongly depends on the amount of interactive feedback. However, important is that *WannaDB* can typically provide high quality with only a few interactions, given that suitable nuggets were extracted in the first place. For most attributes, the first $5 - 10$ interactions massively improve the F1score and achieve gains of up to $0.5$.

#### Runtime & Scalability

The number of interactions needed to achieve a good quality directly influences the time the user has to spend with interaction. As part of the user study described in Section 19.5, we asked real users to use *WannaDB* to extract a subset of the attributes from our datasets (with known gold standard). Figure 5.13 shows the quality achieved by users using the system for the first time, compared to the estimated quality of a simulated user. This experiment confirms that one can achieve high quality table fillings with a low number of interactions. Furthermore, the performance of real users is close to the one of a simulated user. Moreover, our experiments show the real

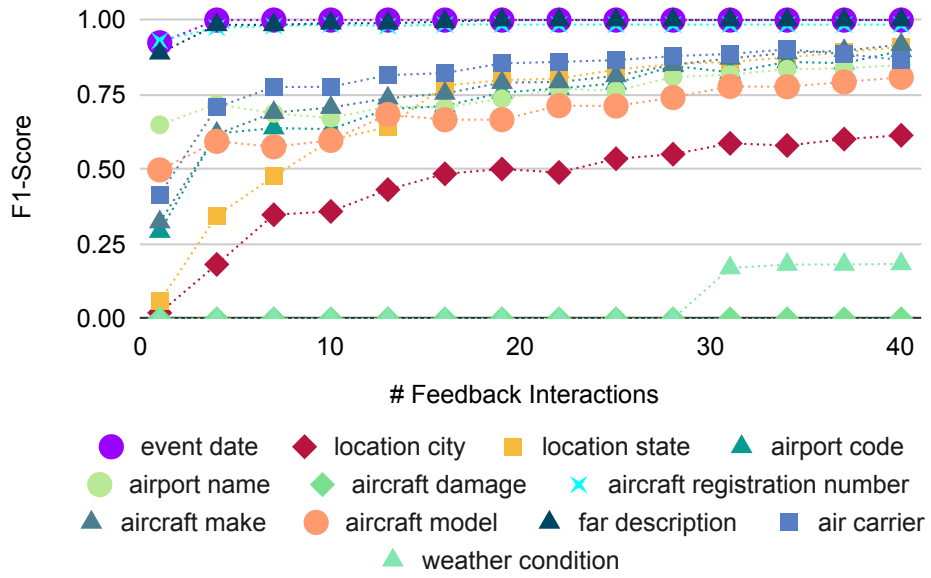



Figure 5.12.: F1 scores of *WannaDB* for the different attributes of the *Aviation* datasets for different amounts of feedback iterations per attribute (1-40). For most attributes, already a small amount of interactions drastically improves the quality, and more interactions lead to continuous improvements. Published in Hättasch et al. [Hät+23a].

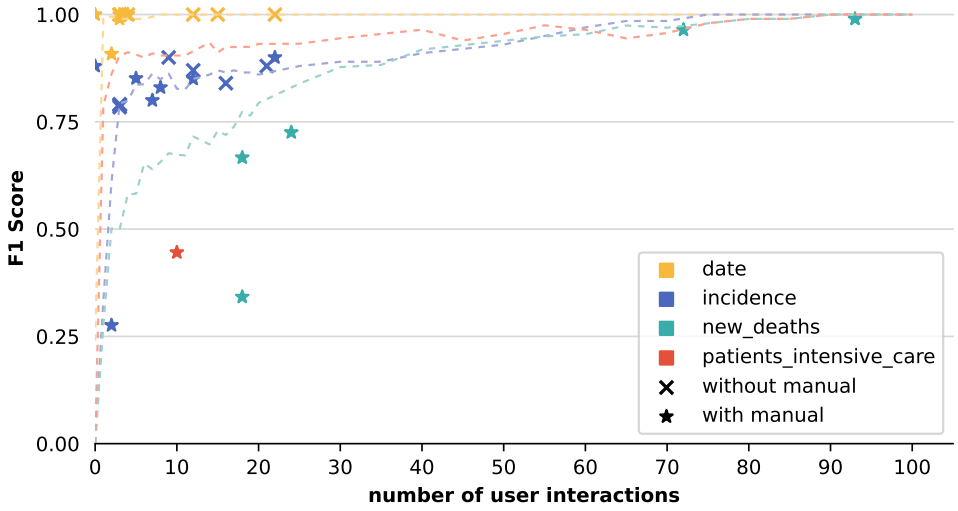



Figure 5.13.: F1 scores of *WannaDB* when used by real users as part of a user study, using the *COVID-19* dataset, dependent on the amount of feedback a user gave for each attribute. The lines indicate the table filling quality achieved by the simulated user feedback. It can be seen that even with a small number of interactions (<10), a high quality can be achieved. The performance by the participants is mostly similar to the one of a simulated user. Published in Hättasch et al. [Hät+23b].

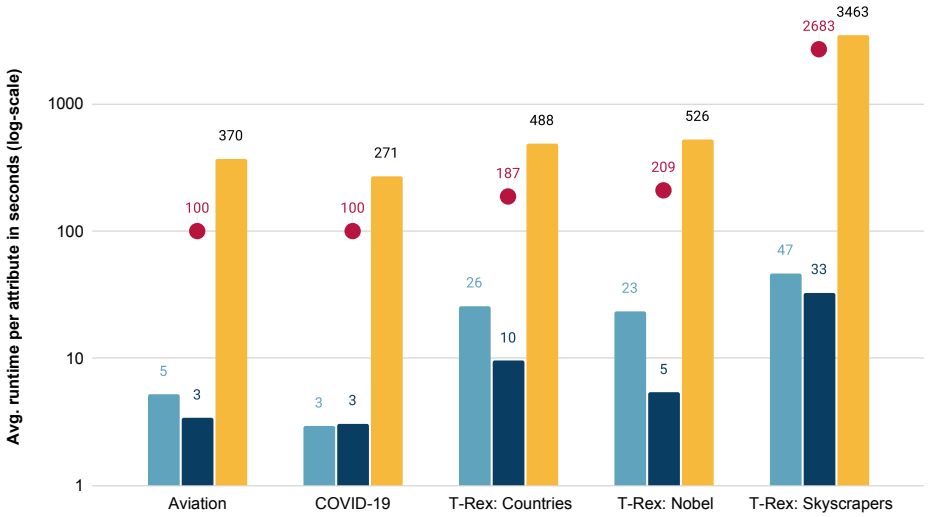



Figure 5.14.: Average runtime per attribute in seconds - logarithmic scale. We compare ■ *WannaDB* on a CPU and ■ *WannaDB* on a consumer GPU with a ■ few-shot BART model on a server GPU. The ● dots show the number of documents per dataset on the same logarithmic scale.
It can be seen that running ■■ *WannaDB* takes orders of magnitudes less time than the ■ BART model. Published in Hättasch et al. [Hät+23b]

.

duration these interactions correspond to: None of our test users needed more than five minutes for up to thirty interactions, two thirds of the attribute fillings were even carried out in two minutes or less. It seems highly unrealistic that a human could perform the same extraction from a hundred documents without technical support in a span even close to this time.

As *WannaDB* is an interactive tool, it is critical to achieve a short latency when processing user interactions and incorporating the feedback to populate the table cells, since too much waiting time for users makes a system uncomfortable to use [LH14; Sch+22]. We therefore aimed at low computation times in between the interactions to keep the users in the loop.

To check whether we reached that aim, we measure the time needed for our system to process 20 user interactions per attribute and apply the feedback to automatically fill the remaining cells. We compare that to the time needed for training and prediction of the few-shot approach described above. For these measurements, we use a consumer machine and GPU. For the few-shot approach described above, we only report the performance for training and prediction on a machine with a GPU designed for machine learning, since the approach would need multiple hours per attribute on a consumer machine. Details, e.g., on the setup, can be found in Section 19.5.1.

Figure 5.14 visualizes the results of our runtime measurements averaged over multiple runs. It can be seen that running *WannaDB* takes orders of magnitudes less time compared to the language model-based BART approach. On our largest dataset *T-Rex: Skyscrapers* (2683 documents), *WannaDB* takes on average 47 seconds on a machine with a CPU only, whereas the BART model nearly takes an hour to predict the cell values from all the 2683 documents, even though it is running on a much more powerful machine with a GPU designed for machine learning. Overall,

our runtime analysis shows that *WannaDB* is indeed usable in an interactive fashion and does not cause long waiting times for the user as a system based on a language model like BART would.

As discussed in Section 5.2.6, our table filling approach has a complexity of $\mathcal{O}(n)$ with regard to the number of nuggets. We can therefore expect our system to scale well with a growing number of documents, which is confirmed by this experiment. Furthermore, the pre-processing stage can be fully parallelized on the document level, again allowing for a good scalability.

## 5.3. Discussion & Future Research

Finding insights in large data collections/lakes is a problem with real-world applications from various domains [e.g., FK14; RV13].

Schema matching can be one method to make sense of the data—by combining different tables. As we showed above, our approach can find relevant correspondences with low overhead, allowing it to scale for large numbers of tables and attributes. Recent studies have shown that prompting approaches can reach high accuracies for schema matching [Nar+22], but as usual with high costs. To improve the natural language understanding (NLU) of our approach without massively increasing the overhead, it would be an interesting future research direction to use foundational models for databases, e.g., as proposed by Vogel, Hilprecht, and Binnig [VHB23] for the representation of table structures and contents while keeping our matching algorithm itself untouched.

For heterogeneous data lakes, information first has to be extracted and organized into a structured representation. Existing systems for extraction are often either domain-specific [e.g., Bri99] or require manual adaption to the application domain [e.g., Caf+07; GB19]. Furthermore, many of them aim at representing a document by a full table that automatically captures its content instead of allowing users to specify their needs [Aro+23]. Moreover, existing low-overhead approaches are often too simple, working mostly only on the syntactical level (e.g., regular expressions, extracting only certain HTML DOM fields). For a more extensive overview of existing approaches and how they relate to our work, we refer to Sections 18.7 and 19.6.

Our approach *WannaDB* provides an alternative to get approximate, high-quality results quickly and at low costs. Our system leverages the advantages of learned language models, in particular their ability to capture linguistic variations, while avoiding costly (domain-specific) fine-tuning and predictions. Furthermore, it does not suffer from the problem of *hallucination* [May+20] that transformer-like models regularly experience, since they aim to also generate values for attributes even if no information nugget is present in the text. *WannaDB* instead generates an empty value in that case.

Recently, large language models such as GPT-3 [Bro+20] with billions of parameters have shown remarkable performance for information extraction tasks. However, there are several ways in which we argue that our approach is better suitable for the exploration of text collections than large language models are: while the pre-processing of text collections with our system is a one time effort and afterwards users can explore the collection freely, large language models need to process all texts of a text collection again whenever the user poses a new query. Due to the size of language models, this takes time and requires substantial computing resources. Currently, large language models provided over APIs have restrictive limits or are only available through pay-per-query offers, which restricts the open-ended exploration of text collections as every query leads to additional

costs. Furthermore, privacy concerns, copyright issues or fear of data leakage might prevent the use of hosted large language models (LLMs). Countering these issues by running a local copy of a large language model is only an alternative when sufficient computing resources are available. In comparison, our system only requires a machine with a consumer GPU for pre-processing once for each new text-collection, the exploration also works on a typical consumer computer without using a GPU (as we have shown before).

However, we believe that our approach can be further improved by leveraging the abilities of LLMs for partial aspects, without processing large parts of the document collection with them. We showed that one shortcoming of our system is that it can only work well when a superset of relevant nuggets was extracted in the preprocessing stage. With the fourth version of our table filling approach (see Section 5.2.6), we introduced a way to add additional nuggets at runtime. While this also tries to find similar new extractions in further documents, the generalization abilities are currently quite basic and mostly on a syntactical level. In future research, we therefore want to evaluate models and approaches that are able to perform additional extractions based on samples given by the user. One way here could be to go in the direction of the recently presented *EVAPORATE-CODE* by Arora et al. [Aro+23], that uses LLMs to generate code for domain specific extractions.

Another interesting research direction would be the integration of data cleaning methods [IC19] into our system. Outlier detection could help to decide whether a (numeric) value seems suitable based upon the other values found for the current attribute. When the value differs greatly, our system could then decide to keep a cell empty or use another value with a similar distance to the confirmed nuggets. Since outlier detection would only be applied as a validation step, approaches with low overhead, e.g., statistical or distance-based outlier detection could be used. Furthermore, our first approaches for semantic grouping could be extended using ideas from data deduplication. To support more complex target schemata (e.g., consisting of multiple tables and allowing to extract multiple lines from a single document), we currently investigate the application of low-overhead generic relation extraction approaches [e.g., ZC21] for the extraction stage that produce additional hints for the interactive table filling.

Finally, the transparency of our system can be further increased by adding visualizations explaining why certain values were selected. These visualizations might even offer additional possibilities for feedback generalization, e.g., through confirming several table rows based upon the closeness to the same confirmed nugget. Therefore, it would be interesting to integrate them into the interactive matching strategy.

# 6. Conclusion

## 6.1. Summary

The necessary information access for many important tasks can only be provided at scale using AI approaches. Yet, systems following most of the existing approaches can only be built and used by IT experts and data scientists. Furthermore, the necessary adaptation and execution causes a high overhead and substantial costs.

A wide application of these approaches can therefore only be achieved by democratizing them by reducing their overhead and improving their usability. In this thesis, we thus presented approaches that bring democratization to three important subfields of data exploration and processing: *Natural Language Interfaces for Data Access & Manipulation*, *Personalized Summarizations of Text Collections*, and *Information Extraction & Integration*.

Our proposed systems all allow for automatization in their fields. Another common property is their efficiency, which allows using them with less data than required by state-of-the-art approaches, or with less computing power required. The systems provide fast results for quick adjustment to the information needs of the users. An additional focus is put on the usability, to not restrict the usage of these tools to people with a strong computer science or data science background.

### Natural Language Interfaces for Data Access & Manipulation

First, we concentrated on natural language interfaces for data access and manipulation. Natural language is a useful alternative interface for relational databases, since it allows users to formulate complex questions without requiring knowledge of SQL. With DBPal (see Section 3.1) we proposed an approach that augments existing deep learning techniques in order to improve the performance of models for natural language to SQL (NL2SQL) translation. It generates synthetic training data adapted to a given database using weak supervision. This allows domain adaption of the model without the need to costly acquire training data (e.g., through annotation by experts or crowd workers). Our evaluation showed that our training pipeline can substantially improve the translation accuracy. Furthermore, we published ParaphraseBench, a benchmark to evaluate how well our and other approaches deal with linguistic variations of the same queries (see Section 3.1.3).

To complement this research, we presented a meta-study on the reproducibility and availability of natural language interfaces for databases (NLIDBs) for real-world applications in Section 3.2. Our analysis led to the conclusion that many NL2SQL approaches are not reproducible. We provided an analysis of reasons and potential ways to counter this in the future.

As the final topic in this research field, we concentrated on natural language interfaces that allow not only information access but also manipulation. These conversational agents (CAs) behave like chatbots and guide a user through an interaction like booking a cinema ticket. We proposed an alternative training pipeline called CAT (see Section 3.3) for creating these CAs with low overhead. While classic approaches require a lot of manually crafted training data, our approach uses weak supervision to generate the training data automatically from a relational database and its set of defined transactions. Our approach is data-aware, i.e., it leverages the data characteristics of the DB at runtime to optimize the dialogue flow and reduce necessary interactions.

### Personalized Summarizations of Text Collections

Our second research area dealt with the use of personalized summaries for the exploration of unknown data collections. The central idea is to produce summaries that exactly cover the current information need of the users. By creating multiple summaries or shifting the focus during the interactive creation process, these summaries can be used to explore the contents of unknown text collections. We proposed Sherlock (see Section 4.1), an approach to create such personalized summaries at interactive speed. To do so, we built on an existing optimization model for the customization, but reduced its runtime from hours per iteration to only seconds. This massive speedup is achieved by sampling. We proposed a method to select the sample size based on iteration time thresholds and evaluated multiple different sampling strategies. Since choosing a suitable sample is not a trivial task, we studied the effectiveness of multiple sampling strategies and the impact of the sample size on the summarization quality. As we showed in our evaluation, our system can provide a similar quality level as the model that is working on the full corpus—at a fraction of runtime.

As part of our research on multi-document summary (MDS), we noticed that there is a lack of diverse evaluation corpora for this task. We therefore presented a framework that can be used to automatically create new summarization corpora and discussed reasonable choices for the parameters. We provided three new sample corpora created with our automatic construction pipeline (FandomCorpora, see Section 4.2). In a comprehensive evaluation, we validated that our pipeline is able to automatically create corpora of use for the training and evaluation of state-of-the-art summarizers.

### Information Extraction & Integration

In the third and last research area, we provided ways to democratize information extraction and integration. This becomes relevant when data is scattered across different sources and there is no tabular representation that already contains all information needed. Therefore, it might be necessary to integrate different structured sources, or to even extract the required information pieces from text collections first and then to organize them.

To integrate existing structured data sources, we presented and evaluated a novel end-to-end approach for schema matching based on neural embeddings in Section 5.1. As part of this, we proposed and analyzed different matchers on multiple levels (i.e., tables and columns) to identify sets of possible table and attribute correspondences. Our results showed that our approach is

able to determine correspondences in a robust and reliable way and can (compared to traditional schema matching approaches) find non-trivial correspondences.

Finally, we tackled the automatic creation of tables from text for situations where no suitable structured source to answer an information need is available. Our proposed approach WannaDB (see Section 5.2) can execute SQL-like queries on text collections in an ad-hoc manner, both to directly extract facts from text documents, and apply filtering, aggregations and grouping. Our system can therefore directly produce tables stating information that is not explicitly mentioned in the documents, and hence not discoverable by pure extraction or search approaches. To do so, we implemented a novel extraction and querying pipeline that first extracts information nuggets independently of the query and then at runtime uses a novel interactive table filling approach that aims to map the information nuggets to a table specified by the user in the form of an attribute list or SQL query. For this table filling, it requests and generalizes feedback from the user. With an extensive evaluation, we showed that our system can indeed fill high-quality tables with only a few user interactions and at interactive speed. A user study confirmed the practical usage for real-world data science scenarios.

In summary, we presented a range of low overhead tools and approaches that allow tackling tasks that could previously only be handled by data scientists. Thus, our approaches democratize the fields of *Natural Language Interfaces for Data Access & Manipulation*, *Personalized Summarizations of Text Collections*, and *Information Extraction & Integration* and are therefore a step towards democratization of information access.

## 6.2. Discussion & Future Research

Overall, we have shown that our systems can provide a quality level comparable to state-of-the-art approaches, but often at a fraction of the associated costs. In fields like the table extractions, we even provide functionality that was—to our knowledge—not covered by any generic tooling available to end users before. Our contributions can be combined into a full pipeline: query existing databases to identify what information is missing, get an overview of the contents of a document collection to check whether it is suitable to augment the database with regard to the information need, fulfill a first approximate extraction and integrate it, then use NLIDBs again to query it and start the cycle over if necessary.

We have focused on low overhead systems, providing this functionality with low associated costs and manual efforts. Yet, since we are working with human language, we need components for natural language understanding (NLU), for semantic and not just syntactic processing. Therefore, to reach sufficient quality, our approaches can be only applied to data in languages where suitable generic resources (like language models (LMs)/embeddings) are available. Luckily, there are nowadays more and more LMs for rare languages and multi-language embeddings [e.g., CC18; LAW21; UR20]. We already evaluated how well our schema matching approach works cross-lingually, and applied our corpus construction framework to German texts. In the future, it would be interesting to adapt and evaluate our other contributions for different languages or even multi-language settings.

As we discussed before, there are some subtasks where prompting large language models (LLMs) outperforms previous approaches (e.g., NL2SQL, paraphrasing, generalization (rules) from examples), yet the high costs and runtimes per request might prevent using these approaches. For future

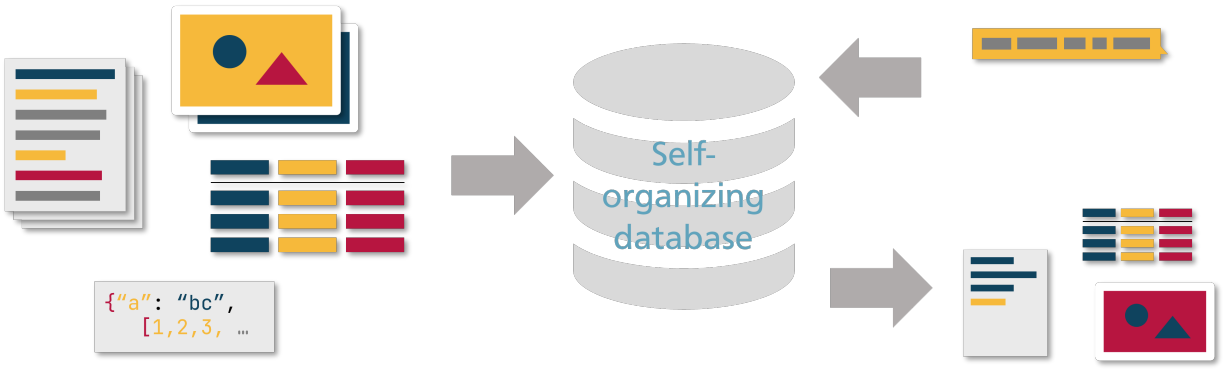



Figure 6.1.: Vision: Self-organizing databases with low overhead

research, it therefore seems reasonable to combine our low overhead approaches with components that use LLMs for certain subtasks, as discussed in Sections 3.4, 4.3, and 5.3.

We are still convinced that many tasks profit from our low overhead approaches with their core properties (local processing without sending data to foreign servers, low computation effort, low manual effort). The best example for this is our extraction system, that allows for exploratory scenarios that would otherwise not be possible at acceptable overhead. Furthermore, compared to the usage of an LLM, it profits from the higher transparency, no hallucinations, and the split between extraction and the computation of aggregation results (instead of prediction).

In Sections 2.1 and 2.6, we described how our tools can be combined to perform multiple actions for data exploration and organization. In the future, we want to go a step further and envision a self-organizing database (see Figure 6.1), where the users can input data in a variety of formats and the system automatically organizes and integrates it based upon the data itself and the queries users pose to it. To build such a system, we want to integrate the components we discussed in this thesis (extraction from text, interactive summarization, NL interfaces, and integration techniques) and enrich them with further components for handling of multi-modal data, output generation, and schema updating.

Moreover, we want to further improve our individual low overhead approaches by integrating ideas from other related fields (e.g., data cleaning, semantic sampling, relation extraction), by leveraging LLMs without substantially increasing the overhead, by applying our approaches to more languages, and by additional work on the usability aspects of our systems. We hope that with this we can again advance democratization of information access.

# Part II.

# Publications

# 7. DBPal: A Learned NL-Interface for Databases (SIGMOD'18)


## Abstract

In this demo, we present DBPal, a novel data exploration tool with a natural language interface. DBPal leverages recent advances in deep models to make query understanding more robust in the following ways: First, DBPal uses novel machine translation models to translate natural language statements to SQL, making the translation process more robust to paraphrasing and linguistic variations. Second, to support the users in phrasing questions without knowing the database schema and the query features, DBPal provides a learned auto-completion model that suggests to users partial query extensions during query formulation and thus helps to write complex queries.


## Bibliographical Information


The content of this chapter was previously published in the peer-reviewed work "Fuat Basik, Benjamin Hättasch, Amir Ilkhechi, Arif Usta, Shekar Ramaswamy, Prasetya Utama, Nathaniel Weir, Carsten Binnig, and Ugur Çetintemel. 'DBPal: A Learned NL-Interface for Databases'. In: *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*. ACM, 2018. DOI: 10.1145/3183713.3193562". The contributions of the author of this dissertation are summarized in Section 3.1.

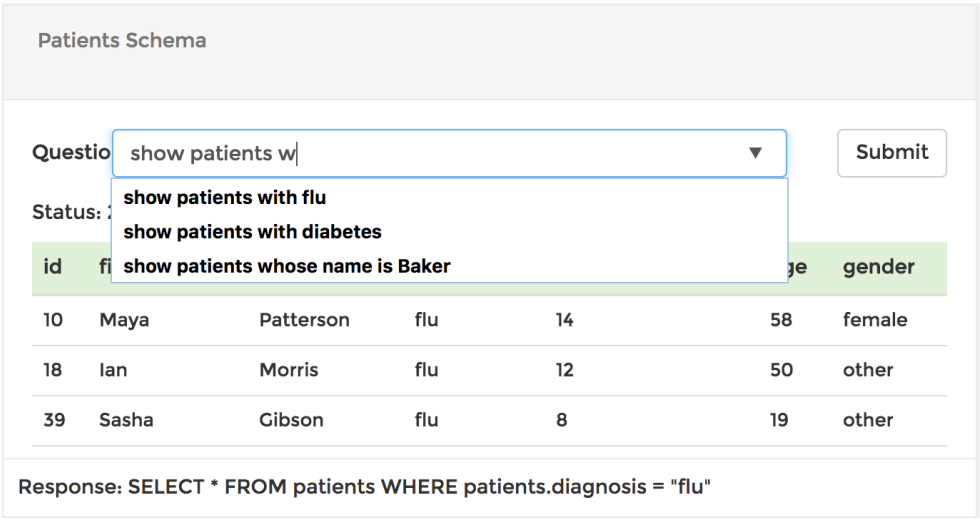



Figure 7.1.: An example session in DBPal

## 7.1. Introduction

Structured Query Language (SQL), despite its expressiveness, may hinder users with little or no relational database knowledge from exploring and making use of the data stored in an RDBMS. In order to effectively leverage their data sets, users are required to have prior knowledge about the schema information of their database, such as table names, columns and relations, as well as a working understanding of the syntax and semantics of SQL. These requirements set "a high bar for entry" for democratized data exploration, and thus have triggered new research efforts to develop alternative interfaces that allow non-technical users to explore and interact with their data conveniently. While visual data exploration tools have recently gained significant attention, Natural language interfaces for databases (NLIDBs) appear as highly promising alternatives since they enable users to pose complex ad-hoc questions in a concise and convenient manner.

For example, imagine that a medical doctor starts her new job at a hospital and wants to find out about the age distribution of patients with the longest stays in the hospital. This question typically requires the doctor – when using a standard database interface directly – to write a complex nested SQL query. Even with a visual exploration tool such as Tableau [Ter+15] or Vizdom [Cro+15a], a query like this is far from being trivial since it requires the user to execute multiple query steps and interactions: the user first needs to find out what "longest stay" means and then use the information to visualize the age distribution for those patients. Alternatively, with an exploration tool supported by a natural language interface, the query would be as simple as stating "What is the age distribution of patients who stayed longest in the hospital?" However, understanding natural language questions and translating them accurately to SQL is a complicated task, and thus NLIDBs have not yet made their way into commercial products.

In this demo paper, we introduce DBPal, a relational database exploration tool that provides an easy-to-use natural language (NL) interface aimed at improving the transparency of the underlying database schema and enhancing the expressiveness and flexibility of human-data interaction through natural language. Different from existing approaches, our system leverages deep models

to provide a more robust query translation. In the following, we outline the two key features of DBPal that are based on deep models.

**Robust Natural-Language-to-SQL Translation:** We propose a robust query translation framework based on the state-of-the-art sequence-to-sequence model for machine translation. Our notion of model robustness is defined as the effectiveness of the translation model to map linguistically varying utterances to finite pre-defined relational database operations. Take, for example, the SQL expression *SELECT * FROM patients WHERE diagnosis='flu'*. There are numerous corresponding natural language utterances for this query, such as *"show all patients with diagnosis of flu"* or simply *"get flu patients"*. We aim to build a translating system that is invariant towards these linguistic alterations, no matter how complex or convoluted. The main challenge hereby is to curate a comprehensive training set. While existing approaches for machine translation require a manually created training set, we designed a novel synthetic training set generation approach that uses only the database schema as input and generates a wide spectrum of pairs of SQL and natural language queries. To further extend the spectrum of natural language variations, we rely on existing language models learned from large text corpora such as Wikipedia to automatically paraphrase and add noise to the training set.

**Interactive Query Auto-Completion:** We provide real-time auto-completion and query suggestion to help users who may be unfamiliar with the database schema or the supported query features. This helps our system improve translation accuracy by leading the user to enter less ambiguous queries. Consider a user exploring a US geographical information database and starting to type *"show me the names "* — at this point, the system suggests possible completions such as *of states*, *of rivers*, or *of cities* to make the user aware of the different options she has. The core of the auto-completion feature is a language model based on the same sequence-to-sequence model and trained on the same dataset as the query translator.

A screenshot of our prototype of DBPal — which implements the aforementioned features — is shown in Figure 7.1. We also recommend the readers to watch the video[1] which shows a recording of a representative user session. The remainder of this paper is organized as follows: We first discuss related prior work in Section 7.2. In Section 7.3, we introduce the system architecture of DBPal. Then, we describe the system demonstration scenario in 7.4. Lastly, in Section 7.5 we discuss some limitations of our current prototype and planned future extensions.

## 7.2. Related Work

NLIDBs have been studied in the database research community since the 1990s [And+95; Pop+03]. Most of this work relied on classical techniques for semantic parsing and used rule-based approaches for the translation into SQL. However, these approaches have commonly shown poor flexibility for the users who use questions with different linguistic styles using paraphrases and thus failed to support complex scenarios.

More recent approaches tackled some of these limitations. For example, the system ATHENA [Sah+16] relies on a manually crafted ontology that is used to make query translation more

[1] https://vimeo.com/user78987383/dbpal

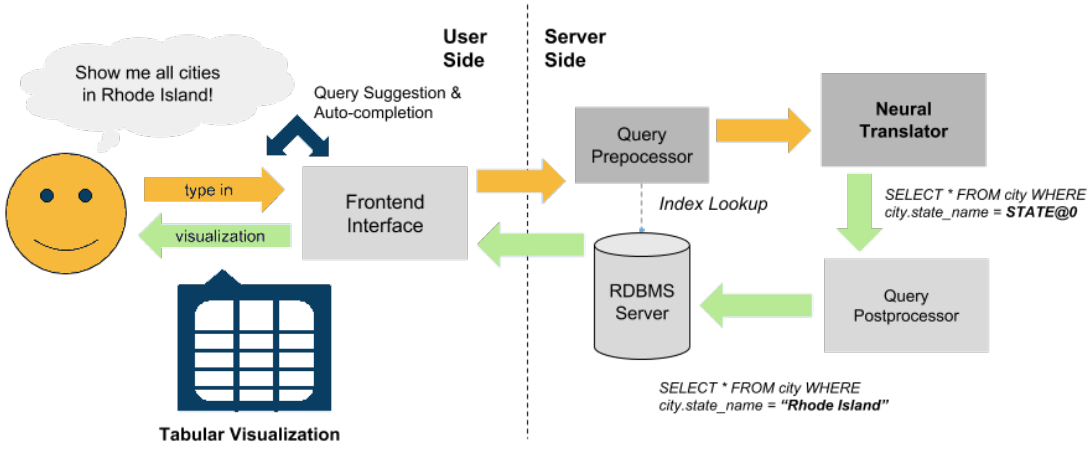



Figure 7.2.: An overview of the DBPal system.

robust by taking different ways of phrasing a query into account. However, since ontologies are domain-specific, they need to be hand-crafted for every new database schema. On the other hand, the system NaLIR [LJ14a] relies on an off-the-shelf dependency parser that could also be built on top of a deep model. However, it still implements a rule-based system that struggles with variations in vocabulary and syntax. Our system attempts to solve both of those issues by being both domain independent and robust to grammatical alterations.

Furthermore, some recent approaches leverage deep models for end-to-end translation similar to our system (e.g., [Iye+17]). However, a main difference between our system and [Iye+17] is that their approach requires manually handcrafting a training set for each novel schema/domain that consist of pairs of natural language and SQL queries. In contrast, our approach does not require a hand-crafted training set. Instead, inspired by [Wan+15], our system generates a synthetic training set that requires only minimal annotations to the database schema. It should be noted that ”no need for manual input” does not mean that our approach does not need training when switching to a schema.

Finally, none of the above-mentioned approaches combine their translation pipelines with additional functions such as auto-completion. These features not only make query formulation easier by helping users to phrase questions even without knowing the database schema, but they also help users to write less ambiguous queries that can be more directly translated into SQL.

## 7.3. System Architecture

The DBPal system consists of three general components: a web-based user interface, a novel query translator, and an interactive auto-completion feature to help users to phrase questions as shown in Figure 7.2. In the following, we first describe the design of each of these components.

### 7.3.1. User Interface Design

We designed a web-based interface that allows users to explore the data in a given database via natural language questions or commands. The interface guides users to perform the exploration in two ways: first, it serves the database schema information and its stored records as tabular views to help users be aware of the accessible information. Second, it applies the auto-completion function as users type in their questions into the form. A primary goal of this input suggestion or auto-completion feature is to steer a user to write an English utterance that is more likely to be correctly translated to SQL by our system. Once the natural language query is submitted, the server translates the query in multiple steps.

### 7.3.2. Natural Language-to-SQL Translation

The main novelty of DBPal is that the query translation from natural language to SQL is modeled as a language translation problem using a state-of-the art Recurrent Neural Network (RNN) model [SVL14]. A major challenge when using such a model is the need for a large and comprehensive training set mapping natural language to SQL. While existing work already has shown that a manually curated training corpus can be used to train a NL-to-SQL sequence-to-sequence model [Iye+17], such an approach imposes a high overhead for every new database schema that needs to be supported.

Our approach is therefore different since it can automatically "generate" a training set for any given database schema automatically as shown in Figure 7.3 (left-hand side). The main idea is that in a first step called *Training Data Instantiation*, we generate a small training set by using a set of simple templates that describe SQL-NL pairs and instantiate them using a given database schema. In the second step called *Training Data Augmentation*, we automatically enrich the simple instantiated SQL-NL pairs by applying different techniques that leverage existing language models to paraphrase but also to introduce noise etc. The goal of the augmentation is to cover a wide spectrum of ways users might ask questions against the given database.

**Training Data Instantiation:** The observation is that SQL – as opposed to natural language – has a much limited expressivity. We therefore use query templates to instantiate different possible SQL queries that a user might phrase against a given database schema such as:

*Select {Att}(s) From {Table} Where {Filter}*

The SQL templates cover a variety of different types of queries from simple select-from-where queries up to more complex aggregate-grouping queries and some simple nested queries.

For each SQL template, we define one or more natural language (NL) templates as counterparts for direct translation such as:

{*SelectPhrase*} *the {Att}(s)* {*FromPhrase*} *{Table}(s)* {*WherePhrase*} *{Filter}*

Reflecting the larger expressivity of spoken language versus SQL, our NL templates contain slots for speech variation (e.g., $SelectPhrase$, $FromPhrase$, $WherePhrase$) in addition to the slots for database items (Tables, Attributes, ...) present in the SQL templates. To instantiate the initial training set, the generator repeatedly instantiates each of our natural language templates by filling

in their slots. Table, column and filter slots are filled using the schema information of the database, while a diverse array of natural language slots is filled using a manually-crafted dictionary of synonymous words and phrases. For example, the $SelectPhrase$ can be instantiated using *What is* or *Show me*. Thus, an instantiated SQL-NL pair might look like this:

*SELECT name FROM patient WHERE age=20* and
*Show me the name of all patients with age 20*

It is also of importance to balance the training data when instantiating the slots with possible values. If we naively replace the slots of a query template with all possible combinations of slot instances (e.g., all attribute combinations of the schema), then instances that result from templates with more slots would dominate the training set and add bias to the translation model. An imbalance of instances can result in a biased training set where the model would prefer certain translations over other ones only due to the fact that certain variations appear more often.

Finally, in the current prototype, for each initial NL template, we additionally provide some manually curated paraphrased NL templates that follow particular paraphrasing techniques as discussed in [VMR11], covering categories such as syntactical, lexical or morphological paraphrasing. Although they are crafted manually, these templates are all database schema independent and can be applied to instantiate NL-SQL pairs for any possible schema without additional manual effort. Moreover, instantiated paraphrased SQL-NL pairs will still be fed into the automatic paraphrasing that is applied next during automatic augmentation.

**Training Data Augmentation:** In order to make the trained deep model more robust, we apply the following post-processing steps after the template instantiation phase:

First, we augment the training set by duplicating NL-SQL pairs, but randomly selecting words of a sentence and paraphrase them using The Paraphrase Database (PPDB) [PC16] as the lexical resource. Second, another problem is missing or implicit information in queries. For example, a user might query the “patients with flu” instead of the “patients diagnosed with flue” and thus the information about the table column might be missing in a user query. To make the translation more robust against missing information, we copy individual NL-SQL pairs, then randomly select words and remove them from the natural language utterances. Third, for each resulting natural language query pair, we lemmatize the words. Finally, constants such as numbers or strings are replaced by special tokens.

In the future, we plan on extending our augmentation techniques; e.g., one avenue is to enhance our automatic paraphrasing using PPDB by paraphrasing multi-word phrases and clauses. We also plan to investigate the idea of using an off-the-shelf part-of-speech tagger to enrich each word in a given query and make the model more robust towards syntactic paraphrasing.

**Training the Model:** We follow a similar architecture of a sequence-to-sequence (seq2seq) model from [SVL14] for machine translation tasks. Our model maps the natural language input directly to SQL output queries, which could be seen as the target language. The model itself consists of two recurrent neural networks, namely, the encoder and decoder. We use a bidirectional encoder-decoder architecture as proposed by [BCB15]. During training, we also apply a dropout on embedding layer to avoid over-fitting to our training corpus, which only consists of a small vocabulary (i.e., SQL keywords as well as schema information of the given database).

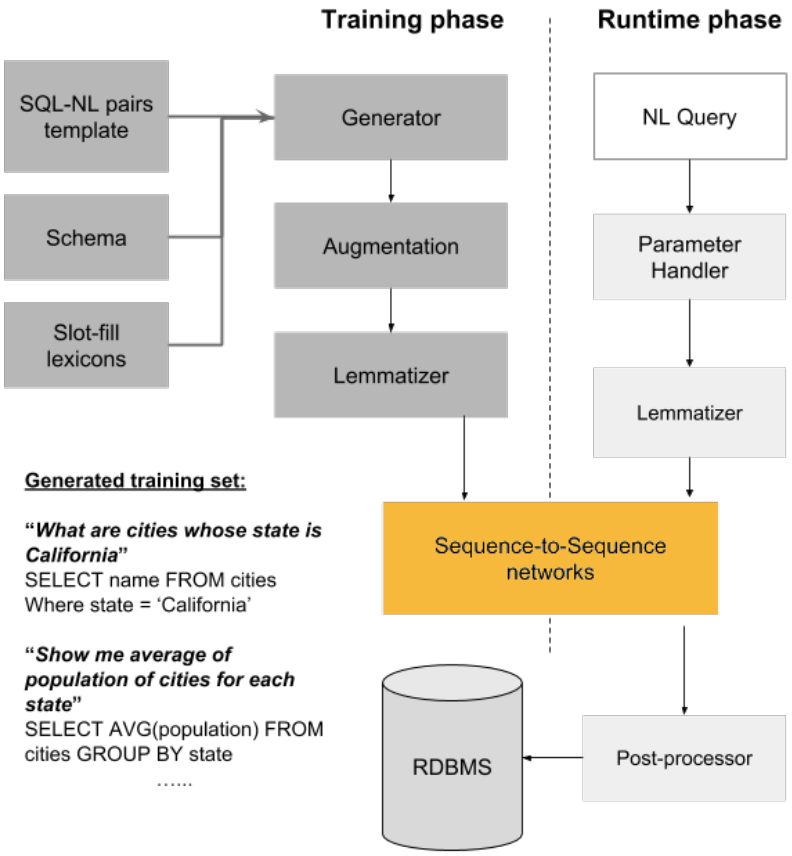



Figure 7.3.: An overview of our Learned Query Translator

**Runtime Usage of Model:** Our learned model is used at runtime to translate an input natural language query coming from a user to an executable SQL query as shown in Figure 7.3 (right-hand side). The runtime pipeline comprises also of multiple steps: First, we handle input parameters (e.g., "New York") and replace them by special tokens (e.g., "CITY_NAME"). The reason is that we want to be able to translate queries for unseen constants correctly. Second, a complex query handler tries to split a natural language query into multiple simple queries that are translatable by our model and combines them using SQL operators such as nesting or joining. Third, after translation, the special tokens are again replaced by their real values in the SQL query in the post-processing phase. For the first two tasks, we rely on another set of deep models. Describing the details of these runtime-models, however, is beyond the scope of the demo paper.

### 7.3.3. Interactive Auto-Completion

To improve the search performance of users, we provide an auto-completion mechanism on queries. This well-known paradigm not only enhances the search behavior, but also helps to increase translation accuracy by leading the user into less ambiguous queries. Considering the abstraction between user and the schema, auto-completion becomes even more crucial. The core of the auto-completion system consists of a generic deep learning model and a modified search algorithm

that gives priority to database specific elements. Given an input sequence, the model performs a breadth-first search in the candidate space, and returns the most probable completions of potential user queries.

## 7.4. Demonstration

The demonstration presents our first prototype of DBPal that allows users to select from two data sets: the ParaphraseBench[2] that consists of a simple one-table dataset of hospital patients and tests the robustness against linguistic variations, and Geo880 (called 'Geo' in this paper), a benchmark dataset of geographical information of the United States that has been commonly tested upon by prior NLIDBs such as [Iye+17] and [Sah+16]. We will allow the user to query both the Patients and Geo datasets with increasingly difficult utterances as shown below. All queries can be formulated with and without the auto-completion feature being enabled.

### 7.4.1. Simple SQL-like Queries

We first provide the user with a list of straightforward functions that line up with the functionality of single-table SQL queries such as simple filter queries and aggregations (e.g. MAX, MIN, COUNT) and GROUP BY statements. We allow the user to play around with the canonical utterances that are syntactically similar to their SQL counterparts for single table in both the Patients and Geo contexts. Example queries include:

- *"show me the distinct names of patients where diagnosis is flu"*
- *"for each gender, what is the minimum length of stay of patients?"*
- *"what is the name and capital of states where name is not Mississippi and population is less than 5000000"*

– *"return the count of mountains where state name is Colorado"*

### 7.4.2. Paraphrased and Fragmented Queries

After working with simple queries, we then allow users to alter their earlier queries to make them less syntactically and semantically aligned with a corresponding SQL query. This can involve restructuring the word syntax of the query, paraphrasing words and phrases with ones that are semantically equivalent, or removing words from the query that are superfluous and/or that do not alter the semantics of the query. Example for this part of the demo are:

- *"distinct names of flu patients"*
- *"find the shortest patient length of stay from each gender"*
- *"enumerate Colorado's mountains"*

[2] `https://link.tuda.systems/paraphrase-bench`

### 7.4.3. Complex Queries

In the last part, we give the user freedom to query the system with whatever utterances they can come up with. These may include contextual questions or more nuanced requests; for example:

- *"Who is the oldest patient?"*
- *"Show the flu patient with the longest length of stay"*
- *"Select the highest point and area from all states"*

## 7.5. Limitation and Future Work

The current prototype of DBPal already shows a significant improvement over other state-of-the-art-systems such as NaLIR [LJ14a] or ATHENA [Sah+16] when dealing with paraphrasing and other linguistic variations. Its main limitation, however, is the lack of coverage for more complicated queries such as different forms of nesting in SQL. However, this is not necessarily a limitation upon the translation model itself, but rather of the training set generation. We are currently extending the training data instantiation phase with additional templates and lexicons as well as the augmentation phase to add more complex natural language queries. Another future avenue of development is allowing users to incrementally build queries in a chatbot-like interface, where the system can ask for clarifications if the model can not translate a given input query directly. We expect that this feature will also be especially helpful for handling nested queries. Finally, integration with other deep models (e.g., for Question-Answering) seems to be another promising avenue for future exploration.

# 8. DBPal: A Fully Pluggable NL2SQL Training Pipeline (SIGMOD'20)


## Abstract

Natural language is a promising alternative interface to DBMSs because it enables non-technical users to formulate complex questions in a more concise manner than SQL. Recently, deep learning has gained traction for translating natural language to SQL, since similar ideas have been successful in the related domain of machine translation. However, the core problem with existing deep learning approaches is that they require an enormous amount of training data in order to provide accurate translations. This training data is extremely expensive to curate, since it generally requires humans to manually annotate natural language examples with the corresponding SQL queries (or vice versa).

Based on these observations, we propose DBPal, a new approach that augments existing deep learning techniques in order to improve the performance of models for natural language to SQL translation. More specifically, we present a novel training pipeline that automatically generates synthetic training data in order to (1) improve overall translation accuracy, (2) increase robustness to linguistic variation, and (3) specialize the model for the target database. As we show, our DBPal training pipeline is able to improve both the accuracy and linguistic robustness of state-of-the-art natural language to SQL translation models.


## Bibliographical Information


The content of this chapter was previously published in the peer-reviewed work "Nathaniel Weir, Prasetya Utama, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Nadja Geisler, Benjamin Hättasch, Steffen Eger, et al. 'DBPal: A Fully Pluggable NL2SQL Training Pipeline'. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. ACM, 2020. DOI: 10.1145/3318464.3380589". The contributions of the author of this dissertation are summarized in Section 3.1.

## 8.1. Introduction

In order to effectively leverage their data, DBMS users are required to not only have prior knowledge about the database schema (e.g., table and column names, entity relationships) but also a working understanding of the syntax and semantics of SQL. Unfortunately, despite its expressiveness, SQL can often hinder non-technical users from exploring and making use of data stored in a DBMS. These requirements set "a high barrier to entry" for data exploration, and have therefore triggered new efforts to develop alternative interfaces that allow non-technical users to explore and interact with their data conveniently.

For example, imagine that a doctor wants to look at the age distribution of patients with the longest stays in a hospital. To answer this question, the doctor would either need to write a complex nested SQL query or work with an analyst to craft the query. Even with a visual exploration tool (e.g., Tableau [Ter+15], Vizdom [Cro+15a]), posing such a query is nontrivial, since it requires the user to perform multiple interactions with an understanding of the nested query semantics. Alternatively, with a natural language (NL) interface, the query is as simple as stating: "What is the age distribution of patients who stayed longest in the hospital?"

Based on this observation, a number of Natural Language Interfaces to Databases (NLIDBs) have been proposed that aim to translate natural language to SQL (NL2SQL). The first category of solutions are rule-based systems (e.g., NaLIR [LJ14a; LJ14b]), which use fixed rules for performing translations. Although effective in specific instances, these approaches are brittle and do not generalize well without substantial additional effort to support new use cases. More recently, deep learning techniques [Iye+17; Wan+15; XLS17] have gained traction for NL2SQL, since similar ideas have achieved success in the related domain of machine translation. For example, generic sequence-to-sequence (seq2seq) [ZXS17] models have been successfully used in practice for NL2SQL translation, and more advanced approaches like SyntaxSQLNet [Yu+18a], which augments deep learning models with a structured model that considers the syntax and semantics of SQL, have also been proposed.

However, a crucial problem with deep learning approaches is that they require an enormous amount of training data in order to build accurate models [HAE16; Sun+17]. The aforementioned approaches have largely ignored this problem and assumed the availability of large, manually-curated training datasets (e.g., using crowdsourcing). In almost all cases, however, gathering and cleaning such data is a substantial undertaking that requires a significant amount of time, effort, and money.

Moreover, existing approaches for NL2SQL translation attempt to build models that generalize to new and unseen databases, yielding performance that is generally decent but does not perform as well as running new queries on the databases used for training. That is, the training data used to translate queries for one specific database, such as queries containing words and phrases pertaining to patients in a hospital, does not always allow the model to generalize to queries in other domains, such as databases of geographical locations or flights.

In order to address these fundamental limitations, we propose DBPal, a fully pluggable NL2SQL training pipeline that can be used with any existing NL2SQL deep learning model to improve translation accuracy. DBPal implements a novel training pipeline for NLIDBs that synthesizes its training data using the principle of *weak supervision* [Cra+00; Deh+17].

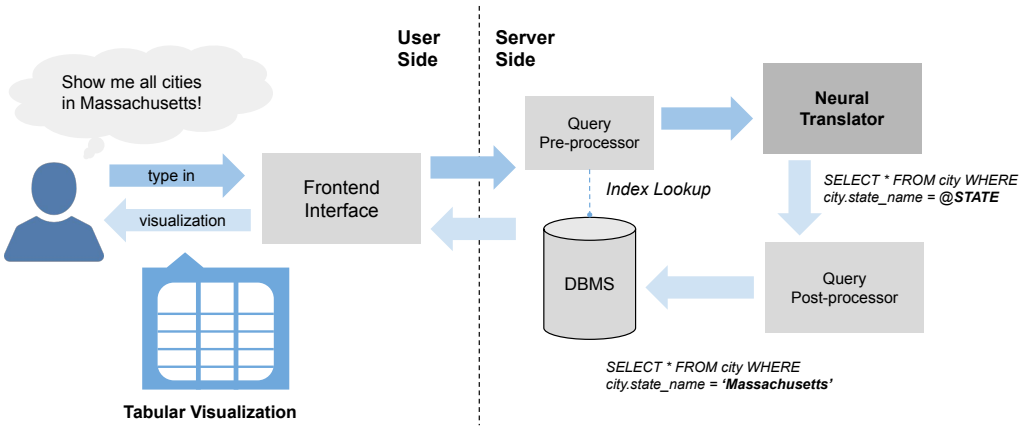



Figure 8.1.: Lifecycle of a NL query through a Neural Translator trained by **DBPal**'s training pipeline

The basic idea of weak supervision is to leverage various heuristics and existing datasets to automatically generate large (and potentially noisy) training data instead of manually handcrafting training examples. In its basic form, only the database schema is required as input to generate a large collection of pairs of NL queries and their corresponding SQL statements that can be used to train any NL2SQL deep learning model.

In order to maximize our coverage across natural linguistic variations, DBPal also uses additional input sources to automatically augment the training data through a variety of techniques. One such augmentation step, for example, is an automatic paraphrasing process using an off-the-shelf paraphrasing database [PC16]. The goal of these augmentation steps is to make the model robust to different linguistic variations of the same question (e.g., "What is the age distribution of patients who stayed longest in the hospital?" and "For patients with the longest hospital stay, what is the distribution of age?").

In our evaluation, we show that DBPal, which requires no manually crafted training data, can effectively improve the performance of a state-of-the-art deep learning model for NL2SQL translation. Our results demonstrate that an NLIDB can be effectively bootstrapped without requiring manual training data for each new database schema or target domain. Furthermore, if manually curated training data is available, such data can still be used to complement our proposed data generation pipeline.

In summary, we make the following contributions:

- We present DBPal, a fully pluggable natural language to SQL (NL2SQL) training pipeline that automatically synthesizes training data in order to improve the translation accuracy of an existing deep learning model.
- We propose several data augmentation techniques that give the model better coverage and make it more robust towards linguistic variation in NL queries.
- We propose a new benchmark that systematically tests the robustness of a NLIDB to different linguistic variations.
- Using a state-of-the-art deep learning model, we show that our training pipeline can improve translation accuracy by up to almost 40%.

The remainder of this paper is organized as follows. First, in Section 8.2, we introduce the overall system architecture of DBPal. Next, in Section 8.3, we describe the details of DBPal's novel training pipeline, which is based on weak supervision. We then show how the learned model for NL2SQL translation is applied at runtime in Section 8.4. Furthermore, we discuss the handling of more complex queries like joins and nested queries in Section 8.5. In order to demonstrate the effectiveness of DBPal, we present the results of our extensive evaluation in Section 8.6. Finally, we discuss related work in Section 8.7 and then conclude in Section 8.8.

## 8.2. Overview

In the following, we first discuss the overall architecture of a NLIDB and then discuss DBPal, our proposed training pipeline based on weak supervision that synthesizes the training data from a given database schema.

### 8.2.1. System Architecture

Figure 8.1 shows an overview of the architecture of our fully functional prototype NLIDB, which consists of multiple components, including a user-interface that allows users to pose NL questions that are automatically translated into SQL. The results from the user's NL query are then returned to the user in an easy-to-read tabular visualization.

At the core of our prototype is a *Neural Translator*, which is trained by DBPal's pipeline, that translates incoming NL queries coming from a user into SQL queries. Importantly, our fully pluggable training pipeline is agnostic to the actual translation model; that is, DBPal is designed to improve the accuracy of existing NL2SQL deep learning models (e.g., SyntaxSQLNet [Yu+18a]) by generating training data for a given database schema.

#### Training Phase

During the training phase, DBPal's training pipeline provides existing NL2SQL deep learning models with large corpora of synthesized training data. This training pipeline, described further in Section 8.2.2, consists of three steps to synthesize the training data: (1) generator, (2) augmentation, and (3) lemmatizer. Once training data is synthesized by DBPal's pipeline, it can then be used (potentially together with existing manually curated training data) to train existing neural translation models that can be plugged into the training pipeline.

#### Runtime Phase

The runtime phase can leverage a model (Neural Translator) that was trained by DBPal, as shown on the right-hand side of Figure 8.2. The *Parameter Handler* is responsible for replacing the constants in the input NL query with placeholders to make the translation model independent from the actual database and help to avoid retraining the model if the underlying database is updated. For example, for the input query shown in Figure 8.2 (i.e., *"What are cities whose state is*

*Massachusetts?"*), the *Parameter Handler* replaces *"Massachusetts"* with the appropriate schema element using the placeholder `@STATE`. The *Lemmatizer* then combines different variants of the same word to a single root. For example, the words *"is"*, *"are"*, and *"am"* are all mapped to the root word *"be"*. Then, the *Neural Translator* works on these anonymized NL input queries and creates output SQL queries, which also contain placeholders. In the example shown in Figure 8.2, the output of the Neural Translator is: `SELECT name FROM cities WHERE state = @STATE`. Finally, the *Post-processor* replaces the placeholders with the actual constants such that the SQL query can be executed.

### 8.2.2. Training Pipeline

The basic flow of the training pipeline is shown on the left-hand side of Figure 8.2. In the following, we describe the training pipeline and focus in particular on the data generation framework. The details of the full training pipeline are explained further in Section 8.3.

#### Generator

In the first step, the *Generator* uses the database schema along with a set of seed templates that describe typical NL-SQL pairs to generate an initial training set. In the second step, *Augmentation*, the training data generation pipeline then automatically adds to the initial training set of NL-SQL pairs by leveraging existing general-purpose data sources and models to linguistically modify the NL part of each pair.

The main idea is that each seed template covers a typical class of SQL queries (e.g., a `SELECT-FROM-WHERE` query with a simple predicate). Composing the seed templates is only a minimal, one-time overhead, and all templates are independent of the target database (i.e., they can be reused for other schemas). Furthermore, in DBPal, we assume that the database schema provides human-understandable table and attribute names, but the user can optionally annotate the schema to provide more readable names if required; deriving readable schema names automatically is an orthogonal issue.

The schema information is then used to instantiate these templates using table and attribute names. Additionally, manually predefined dictionaries (e.g., to cover synonyms) can be used to instantiate simple variations of NL words and phrases (e.g., "Show me" and "What is" for the `SELECT` clause). Currently, DBPal contains approximately 100 seed templates. A typical training set that can be generated from these templates contains around 1 million NL-SQL pairs for a simple, single-table database schema and around 2-3 million for more complicated schemas.

#### Augmentation

A core aspect of our pipeline is the *Augmentation* step that automatically expands the training data produced by our Generator in order to offer more accurate and linguistically robust translations. During augmentation, the training data generation pipeline automatically adds new NL-SQL pairs by leveraging existing general-purpose data sources and models to linguistically vary the NL part of each pair. The goal of the augmentation phase is thus to cover a wide spectrum of linguistic variations for the same SQL query, which represent different versions of how users might phrase

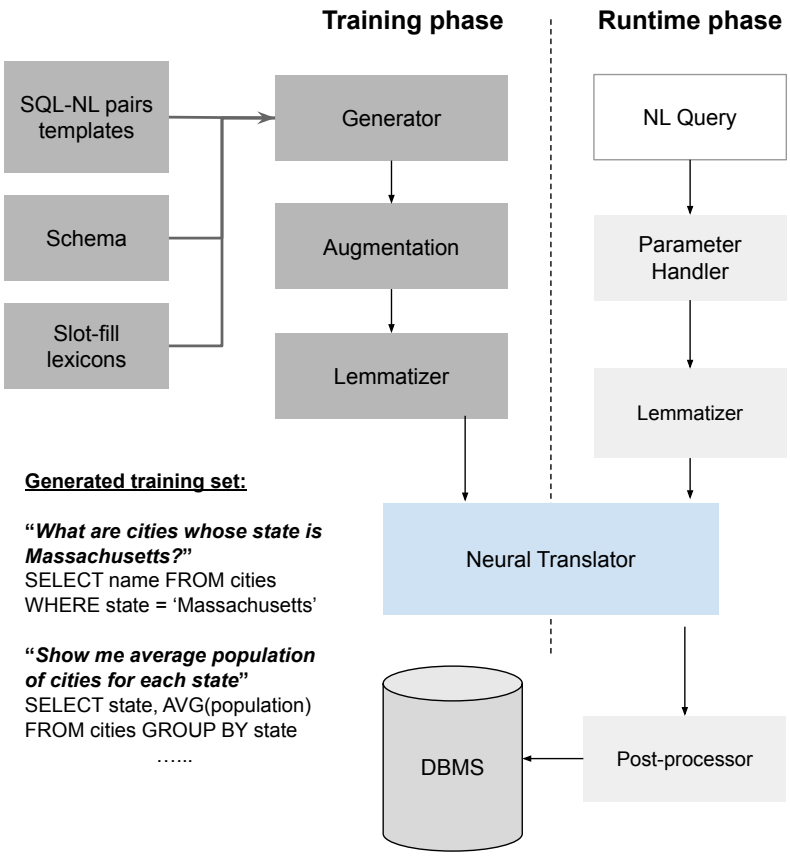



Figure 8.2.: **DBPal**'s Training and Runtime Phases

the query in NL. This augmentation is the key to make the translation model robust and allows DBPal to provide better query understanding capabilities than existing standalone approaches. Section 8.3.2 describes this process in more detail.

#### Lemmatization

Finally, in the last step of the data generation procedure, the resulting NL-SQL pairs are lemmatized to normalize the representation of individual words. During this process, different forms of the same word are mapped to the word's root in order to simplify the analysis (e.g., *"cars"* and *"car's"* are replaced with *"car"*). The same lemmatization is applied at runtime during the aforementioned pre-processing step.

## 8.3. Training Phase

In this section, we describe DBPal, our fully pluggable training data generation pipeline, which is designed to improve the translation accuracy and linguistic robustness of existing NL2SQL

deep learning models. After describing the steps of our training pipeline in detail, we discuss an optimization procedure of the data generation process to increase the model quality via parameter tuning. Finally, we elaborate on the model training process, including a description of the details of the model architecture and the hyperparameters used in training.

### 8.3.1. Data Instantiation

The main observation of the instantiation step is that SQL, as opposed to NL, has significantly less expressivity. We therefore use query templates to instantiate different possible SQL queries that a user might phrase against a given database schema, such as:

*Select {Attribute}(s) From {Table} Where {Filter}*

The main idea of data instantiation is that the space of possible SQL queries a user might phrase against a given database schema can be defined using a set of SQL templates. The SQL templates cover a variety of query types, from simple `SELECT-FROM-WHERE` queries to more complex group-by aggregation queries, as well as some simple nested queries. For each SQL template, we define one or more NL templates as counterparts for direct translation, such as:

{*SelectPhrase*} *{Attribute}(s)* {*FromPhrase*} *{Table}(s)* {*WherePhrase*} *{Filter}*

It is important to note that we do not use actual constants in the filter predicates. Instead, we use placeholders (e.g., $@AGE$) that represent an arbitrary constant for a given table attribute. This makes the model trained on the generated data independent of concrete values used in the database; thus retraining is not required after inserts or updates.

To account for the expressivity of NL compared to SQL, our templates contain slots for speech variation (e.g., *SelectPhrase*, *FromPhrase*, *WherePhrase*) in addition to slots for database objects (e.g., tables, attributes). Then, to instantiate the initial training set, the Generator repeatedly instantiates each of our NL templates by filling in the corresponding slots. Table, column, and filter slots are filled using information from the database's schema, while a diverse array of NL slots are filled using manually crafted dictionaries of synonymous words and phrases. For example, the phrases *"what is"* or *"show me"* can be used to instantiate the *SelectPhrase*. A fully instantiated NL-SQL pair might look like:

| **Instantiated NL:** | **Instantiated SQL:** |
|---|---|
| *Show the names of all patients with age 20.* | `SELECT name FROM patients WHERE age = 20` |

An important part of training data instantiation is balancing the number of NL-SQL pairs that are instantiated per template. If we naively replace the slots of a query template with all possible combinations of slot instances (e.g., all attribute combinations from the schema), then instances that result from templates with more slots would dominate the training set and bias the model. More specifically, an imbalance of instances can lead to a biased training set where the model would prefer certain translations over others only due to the fact that certain variations appear more often. We therefore randomly sample from the possible instances to get a good coverage of different queries and to keep the number of instances per query template balanced.

Finally, for each initial NL template, we additionally provide some manually curated paraphrased NL templates that follow particular paraphrasing techniques [VMR11], covering categories such as

syntactical, lexical, and morphological paraphrasing. The existence of multiple corresponding NL templates for each SQL template allows us to systematically cover a range of possible linguistic variations. Importantly, the paraphrased templates can be applied to instantiate the training data for any given schema, and the instantiated NL-SQL pairs are also automatically paraphrased during automatic data augmentation.

For example, consider the following paraphrased NL template and a corresponding instantiation:

| **Paraphrased NL Template:** | **Instantiated NL Template:** |
|---|---|
| *For {Table}(s) with {Filter}, what is their {Attribute}(s)?* | *For patients with age @AGE, what is their name* |

We further expand upon our augmentation techniques in the following section.

### 8.3.2. Data Augmentation

Unsurprisingly, numerous ways exist to express the same idea in NL. For example, the questions *"Show me the names of all patients older than 18"* and *"What are the names of patients who have an age greater than 18?"* are semantically equivalent. Therefore, to make the NL2SQL model more robust to these linguistic variations, we apply the following augmentation steps for each instantiated NL-SQL pair.

#### Automatic Paraphrasing

First, we augment the training set by generating duplicate NL-SQL pairs. This process involves randomly selecting words/subphrases of the NL query and paraphrasing them using the Paraphrase Database (PPDB) [PC16] as a lexical resource, for example:

**Input NL Query:**
*Show the names of all patients with age @AGE.*

**PPDB Output:**
demonstrate, showcase, display, indicate, lay

**Paraphrased NL Query:**
*Display the names of all patients with age @AGE.*

PPDB is an automatically extracted database containing millions of paraphrases in 27 different languages. DBPal uses PPDB's English corpus, which provides over 220 million paraphrase pairs consisting of 73 million phrasal and 8 million lexical paraphrases, as well as 140 million paraphrase patterns, which capture a wide range of meaning-preserving syntactic transformations. The paraphrases are extracted from bilingual parallel corpora totaling over 100 million sentence pairs and over 2 billion English words.

During paraphrasing, we randomly replace words and subphrases of the input NL query with available paraphrases provided by PPDB. For example, searching in PPDB for a paraphrase of the word *enumerate*, as in *"Enumerate the names of patients with age 80"*, we get suggestions such as *"list"* or *"identify"* as alternatives.

An important question is how aggressively to apply automatic paraphrasing. We therefore provide two parameters to tune the automatic paraphrasing in DBPal. The first parameter $size_{para}$ defines the maximum size of the subclauses (in number of words) that should be replaced in a given NL query. A second parameter $num_{para}$ defines the maximum number of paraphrases that are generated as linguistic variations for each subclause. For example, setting $size_{para} = 2$ will replace subclauses of size $1$ and $2$ (i.e., unigrams and bigrams) in the input NL query with paraphrases found in PPDB. Furthermore, setting $num_{para} = 3$, each of the unigrams and bigrams will be replaced by at most $3$ paraphrases.

Setting these two parameters in an optimal manner is, however, not trivial: if we set both parameters to high values, we can heavily expand our initial training set of NL-SQL query pairs using many different linguistic variations, which hopefully will increase the overall robustness of DBPal. At the same time, we might also introduce noise into the training dataset, since PPDB also includes some paraphrases that are of low quality.

DBPal has default values for all of these parameters that we have empirically determined to have the best performance on multiple database schemas, which are used in our evaluation in Section 8.6. In order to optimize these parameters to increase the overall model accuracy for a given input database schema, we provide an optimization procedure that we discuss in Section 8.3.3. Our procedure automatically finds a parameterization of the data generator that balances, among others, the trade-off between these two dimensions: size of the augmented training data and noise in the training data.

#### Missing Information

Another challenge of input NL queries is missing or implicit information. For example, a user might ask for *"patients with influenza"* instead of *"patients diagnosed with influenza"*, where the referenced attribute (i.e., *"diagnosis"*) is never explicitly stated.

Therefore, to make the translation more robust to missing or implicit context, we randomly drop words and subphrases from the NL training queries. For example, from the sentence *"patients diagnosed with influenza"*, DBPal might decide to drop the word *"diagnosed"*, allowing the translation model to be able to successfully answer the question *"Who are the patients with influenza?"*

Similar to paraphrasing, an interesting question is: which words or subphrases should be removed and how frequently should we remove them? Again, aggressively removing words increases the training data size, since more variations are generated. On the other hand, however, we might introduce noisy training data that leads to a drop in translation accuracy and, counterproductively, produces less linguistically robust models.

In order to tune how aggressively we drop words and subphrases, we follow a similar protocol as the paraphrasing process by randomly selecting words in the NL query and removing them in a duplicate. Thus, we additionally introduce a parameter named $num_{missing}$ that defines the maximum number of query duplicates with removed words for a given input NL query. We also include a parameter $randDrop_p$ that defines how often the generator will choose to remove words from a particular NL query at all. Analogously to automatic paraphrasing, we set these two parameters for a given input database schema automatically using the procedure described in Section 8.3.3.

#### Other Augmentations

For the automatic data augmentation, we apply some additional techniques to increase the linguistic variations. One example is the use of available linguistic dictionaries for comparatives and superlatives. For example, by using these resources, we can replace the general phrase *greater than* in an input NL query by *older than* if the domain of the schema attribute is set to *age*.

In the future, we plan on extending our augmentation techniques in a variety of ways. For example, one possible avenue is to enhance our automatic paraphrasing using other language sources and not only PPDB. We also plan to investigate the idea of using an off-the-shelf part-of-speech tagger to annotate each word in a given NL query. These annotations can be used in different forms (e.g., we could use them in the automatic paraphrasing to identify better paraphrases or to infer a request for a nested query). Another idea is to use part-of-speech tags to apply the word removal only for certain classes of words.

### 8.3.3. Optimization Procedure

One important challenge of the automatic data generation steps is to instantiate the training data such that the translation model will provide a high accuracy. For example, the template-based training data instantiation step also has parameters that can be tuned to control the number of basic NL-SQL pairs that are instantiated for each template. Without tuning these parameters, the data generation process could introduce bias in the generated training data based on a given schema if we exhaustively generate all possible NL-SQL pairs. Furthermore, the augmentation steps require several parameters for each step that define how aggressively paraphrasing and removing information is applied to an input NL query.

We therefore attempt to automatically optimize the configuration of the generator parameters given a particular database schema. The intuition behind this strategy comes from observations made about the translation model's behavior. In particular, we note that models are typically very susceptible to overfitting to over-represented NL-SQL queries. For example, if we overpopulate the training set with the SQL *count* queries (the natural language parallel will usually include words like *"how many"*), the model will likely output a *count* query for all aggregations simply because it sees particular NL words that most commonly appeared with the word *count* during training. Queries like *"How large is the area of Alaska?"* might be therefore be mapped to a *count* instead of *sum* simply for this reason.

Table 8.1 lists all parameters that are available in DBPal to tune the data generation process and explains their meanings. As mentioned before, these parameters define the main characteristics of the training data instantiation and augmentation steps, and thus they have an effect on the accuracy and robustness of the translation model. In order to find the optimal parameter values of the data generation process for a given schema automatically, we model the data generation procedure as the following function:

$$Acc = Generate(D, T, \phi)$$

The inputs of this function are the database $D$ that describes the schema and contains some sample data, a test workload $T$ of input NL queries and expected output SQL queries, as well as a possible

| Parameter | Explanation |
|---|---|
| **Data Instantiation** | |
| $size_{slotfills}$ | Maximum number of instances created for a NL-SQL template pair using slot-filling dictionaries. |
| $size_{tables}$ | Maximum number of tables supported in join queries. |
| $groupBy_p$ | Probabilities of generating a *GROUP BY* version of a generated query pair. |
| $join_{boost}$, $agg_{boost}$, $nest_{boost}$ | Control the balance of various types of SQL statements relative to each other and the number of templates used. |
| **Data Augmentation** | |
| $size_{para}$ | Maximum size of subclauses that are automatically replaced by a paraphrase. |
| $num_{para}$ | Maximum number of paraphrases that are used to vary a subclause. |
| $num_{missing}$ | Maximum number of words that are removed for a given input NL query. |
| $randDrop_p$ | Probability of randomly dropping words from a generated query. |

Table 8.1.: Parameters of the Data Generation Procedure

instantiation of all the tuning parameters $\phi$ listed in the Table 8.1. The output of the generation procedure $Acc$ is the accuracy of our model that is trained on the generated dataset using $D$ as well as $\phi$ and then evaluated using the test workload $T$. It is important to note that we can either use a test workload $T$ that is created automatically by using a random sample of the generated training data (i.e., we split the test set from the training set) or by providing a small representative set of NL-SQL query pairs that are curated manually.

The goal of the optimization procedure is to find a parameter set $\phi$ that maximizes the accuracy $Acc$. Automatic optimization techniques are useful for global optimization problems that evaluate expensive black-box functions; as such it has become popular for optimizing deep learning models that take in a seemingly arbitrary set of hyperparameters, such as the number of layers or perceptrons per layer of a convolutional neural network (CNN). However, instead of applying the optimization procedure to our translation model, we extrapolate one step backwards and attempt to optimize the nature of the training set to which the model will be exposed.

In machine learning, there exist several strategies for automatically tuning hyperparameters. In DBPal, we use a random search approach to automatically tune the hyperparameters $\phi$ of the function $Generate$. For each candidate set of parameters, the entire system pipeline, including data generation and model training (labeled *Generate*$(D, T, \phi)$), is completed and the accuracy is returned. Random search is a standard technique for hyperparameter-tuning and differs from grid search, which is an alternative to random search, mainly in that it searches the specified subset of hyperparameters randomly instead of exhaustively.

The major benefit of random search is the reduced runtime in practice to find a set of hyperparameters that increases the accuracy of the learned model. However, unlike grid search, with random search we are not guaranteed to find the optimal combination of hyperparameters. In the experimental evaluation, we show that by using random search, we can find parameters for the data generation process to produce training data that can provide a high accuracy for the trained model. We also experimented with more sophisticated hyperparameter search strategies

like Bayesian optimization, which did not find to improve the accuracy over the random search strategy.

### 8.3.4. Neural Translation Model

As previously mentioned, DBPal is fully pluggable and is designed to improve the accuracy of any existing NL2SQL deep learning model. Therefore, importantly, existing models, ranging from simple seq2seq to more complex ones like SyntaxSQLNet [Yu+18a], can be used for the translation and still benefit from our proposed training pipeline. Additionally, since a great deal of ongoing work is currently focused on producing better NL2SQL models, our approach is similarly able to improve the performance of any new advancements that the NL community develops for translation. Since our main contribution of this work is the novel data generation approach, a detailed discussion of deep model architectures is beyond the scope of this paper.

## 8.4. Runtime Phase

In this section, we describe the query translation pipeline. The complete process of the runtime phase is shown in Figure 8.2 (right-hand side). From the given input NL query to the output SQL query, three major processing phases are performed: pre-processing, query translation, and post-processing. The output SQL query is then executed against the database and the result is returned to the user interface in tabular form, as shown in Figure 8.1.

### 8.4.1. Pre-Processing and Query Translation

The input to the pre-processing step is a NL query formulated by the user, such as the following:

**User NL Query (with constants):**
*Show me the name of all patients with age 80*

As previously mentioned, during pre-processing, parameter values (i.e., constants) are replaced with special placeholders. This step is performed to translate queries independently of the database content. The resulting intermediate query is as follows:

| **Input NL Query (without constants):** | **Output SQL Query (without constants):** |
|---|---|
| *Show me the name of all patients with age @AGE* | `SELECT name FROM patient WHERE age=@AGE` |

Replacing the constants in the input NL query with their placeholders is a nontrivial task. The process might not be deterministic, since the same constant might map to different columns. This sub-task, commonly referred to as "variable anonymization," has been identified by other groups as an important challenge in the NL2SQL pipeline. In their work towards systematic benchmarking for NL2SQL systems, [Fin+18] acknowledge that anonymization can be treated as a separate task, and provide benchmarks with and without having already performed the anonymization. As such, our paper follows the former setup and evaluates on test sets with pre-anonymized values.

In practice, as a temporary solution in the basic version of DBPal, we build an index on each attribute of the schema that maps constants to possible attribute names. Moreover, the user might have provided a string constant in the input NL query that is only similar to the one used in the database (e.g., the user provides "New York City" instead of "NYC"). In the current version of DBPal, we use a similarity function to replace constants with their most similar value that is used in the database. We therefore search the possible column values and compute a string similarity metric with the string constant provided by the user. In our prototype, we currently use the Jaccard index, but the function can be replaced with any other similarity metric. In cases where the similarity of all values for the user-specified string is too low (which could mean that the value does not exist in the database), we use the constant as given by the user and do not replace it.

Finally, as a last step of pre-processing, we lemmatize the input NL query to normalize individual words and thus increase the similarity of the training data (which we also lemmatize) and the input NL query the user provides at runtime. After all pre-processing steps are applied, the trained model is used to map the anonymized and lemmatized NL query into an output SQL query, as shown previously.

### 8.4.2. Post-Processing

After pre-processing and translation, a post-processing phase is applied. First, the placeholders in the output SQL query are replaced by the appropriate database constants. Then, we use SQL syntax knowledge to repair potential translation errors of the model.

The first step is simply the inverse step of the pre-processing phase. For example, the placeholder in the output SQL query shown before should be replaced by the according constant that was present in the user input:

**Output SQL Query (with constants):**
```
SELECT name FROM patient WHERE age=80
```

Hence, we need to replace the placeholder in the SQL output of the model with the constant used in the input NL query (e.g., @AGE is replaced by $80$ in the example above).

In the second step of the post-processing phase, DBPal uses knowledge about the SQL syntax to repair potential translation errors that might result from applying the model. One typical example is that the attributes used in the output SQL query and the table names do not match (e.g., the query asks for patient names but the table patient is not used in the `FROM` clause). In this case, the post-processing step adds missing tables to the `FROM` clause. The most likely join path is selected from the schema using the shortest join path between the table already present in the `FROM` clause and the missing table. This is similar to the general join handling, which we discuss in detail in the next section.

## 8.5. Complex Queries

In the previous sections, we have shown both the training and runtime phase of DBPal for example queries with single tables. In this section, we discuss how we extend these techniques to handle joins and nested queries as well.

### 8.5.1. Join Queries

In order to handle NL input queries that require a join, we extend the template-based instantiation during the training phase such that the attribute slots of a query can be filled with attribute names from multiple tables in the same instance. Attribute slots can be present in different parts of a query (e.g., the `SELECT` or `WHERE` clause). The maximum number of distinct tables that are used during slot-filling can be defined using a parameter called $size_{tables}$, which is a tuning parameter of the data generation process, as previously discussed. Furthermore, we also change the instantiation of table names in the generated SQL query. Instead of enumerating all required tables in the `FROM` clause, we add a special placeholder *@JOIN*. An example for an instantiated NL-SQL pair that use a join might look as follows:

| **SQL Query (Training Set):** | **NL Query (Training Set):** |
|---|---|
| `SELECT AVG(patient.age) FROM @JOIN WHERE doctor.name=@DOCTOR.NAME` | *What is the average age of patients treated by doctor @DOCTOR.NAME* |

At runtime, our translation model then outputs a SQL query with a *@JOIN* placeholder when it sees an input NL query with attributes from multiple tables (i.e., it outputs a SQL query without concrete table names in the `FROM` clause). The *@JOIN* placeholder is then replaced in the post-processing step with the actual table names and the join path that contains all tables required by the query. From experience, we observe that this reduces the overall model complexity, since the model does not need to predict actual table names for the `FROM` clause.

Furthermore, as explained before in Section 8.4, for single-table queries our translation model sometimes produces erroneous SQL queries where the table name in the `FROM` clause does not match the attribute names used. These errors are handled in the post-processing step, where we must infer the correct table names from the attributes used in the SQL query. Thus, increasing the model complexity to predict both the join paths and table names increases the rate of errors that would need to be handled in the post-processing phase. The introduction of the `JOIN` placeholder rectifies these issues.

DBPal's post-processing phase uses the schema information to infer table names and a join path from the attributes in the SQL output of the model. In case multiple join paths are possible to connect all the required tables, we select the join path that is minimal in its length.

### 8.5.2. Nested Queries

Handling arbitrary nested queries is a hard task on its own. In our current prototype, we only handle a subset of possible SQL nestings by adding additional templates that represent common forms of nested queries where the slots for the outer and inner query can be instantiated individually. An example for a NL-SQL template pair looks as follows:

| **SQL Template:** | **NL Template:** |
|---|---|
| *Select {Attribute}(s) From {Table} Where (Select {MaxMinAttribute} From {Table} Where {Filter}))* | *{SelectPhrase} the {Attribute}(s) {FromPhrase} {Table} {WherePhrase} {MaxMinAttribute}* |

| Algorithm | Easy | Medium | Hard | Very Hard | Overall |
|---|---|---|---|---|---|
| SyntaxSQLNet | 0.445 | 0.227 | 0.231 | 0.051 | 0.248 |
| DBPal (Train) | 0.472 | 0.300 | 0.252 | 0.107 | 0.299 |
| DBPal (Full) | 0.480 | 0.323 | 0.279 | 0.122 | 0.317 |

Table 8.2.: Spider Benchmark Results

This template is then instantiated during the first phase of the data generation process. For example, the following pair of instantiated queries could be generated for the training set from the previous template pair:

**SQL Query (Training Set):**
```
SELECT name FROM mountain WHERE
height = (SELECT MAX(height) FROM
mountain WHERE state=@STATE.NAME)
```

**NL Query (Training Set):**
*What is name of the mountain with maximum height in @STATE.NAME*

The instantiated queries are augmented automatically in the same way as for non-nested queries. In its current version, DBPal only supports uncorrelated nestings in the `WHERE` clause using different keywords (e.g., `EXISTS`, `IN`), as well as nested queries where the inner query returns an aggregate result. However, the nesting capabilities of DBPal can easily be extended by further adding templates that are instantiated in the first phase of the data generation.

## 8.6. Experimental Evaluation

The main goal of our evaluation is to show that the presented training pipeline is able to improve the performance of existing NL2SQL translation techniques. Therefore, in Section 8.6.1, we first compare our proposed augmentation techniques to the training process using SyntaxSQLNet [Yu+18a] with the well-known Spider [Yu+18b] benchmark. Based on this analysis, in Section 8.6.2, we introduce a new benchmark for NLIDBs that better tests linguistic variations for NL2SQL translation and present experimental results. Finally, Section 8.6.3 presents the results of several microbenchmarks that test different aspects of DBPal's training pipeline.

### 8.6.1. Existing Benchmark: Spider

The first benchmark that we use to show the effectiveness of our proposed techniques is Spider [Yu+18b]. In the following, we describe the benchmark at a high-level, and then we show how DBPal can effectively improve the accuracy of SyntaxSQLNet [Yu+18a] on the Spider benchmark. SyntaxSQLNet is a state-of-the-art deep learning model that uses pre-trained GloVe word embeddings [PSM14] when parsing the words in the input sentences. Using GloVe embeddings already allows the model to handle variations of individual words efficiently.

#### Setup

Spider [Yu+18b] is a popular openly-available dataset that consists of over $10,000$ NL questions paired with the corresponding SQL queries. The benchmark contains $200$ database schemas, each of

which has several tables, representing real-world database deployments. The data in the benchmark is very diverse and spans $138$ distinct domains (e.g., automotive, social networking, geography). In addition to the diverse data, the corresponding SQL queries contain almost all of the common SQL patterns, including nested queries.

Based on the complexity of the corresponding SQL query (i.e., the number of SQL components), each question is assigned a difficulty (i.e., easy, medium, hard, very hard). The benchmark includes queries from each of these categories, allowing us to test how different approaches compare in a diverse set of scenarios. In this benchmark, accuracy is measured by computing the number of correctly translated NL phrases divided by the total number of queries. A query is deemed to be correctly translated only if it exactly matches the provided "gold standard" SQL query for the NL input, without allowing for semantically equivalent answers.

Unlike existing datasets, Spider uses different databases (i.e., schemas and data) for training and testing (i.e., a database schema is used exclusively for either training or testing, but not both). This allows us to evaluate how well the model will generalize to new domains.

### Results

Table 8.2 shows the accuracy for SyntaxSQLNet using the Spider dataset for three different configurations. First, as a baseline, we show the performance of the base SyntaxSQLNet model trained using the data from Spider's training set. The DBPal (Train) configuration uses the baseline SyntaxSQLNet (i.e., trained using Spider's training set), but we augmented the training data with additional synthetic data generated by DBPal using the schemas of the training set in Spider only. Finally, the DBPal (Full) version uses the schemas from both the training and test set of Spider to generate additional synthetic training data. Note, however, that DBPal never sees actual NL-SQL pairs from the test set during the training process, only the schemas in the DBPal (Full) configuration.

As shown, both configurations of DBPal improve upon the baseline performance of SyntaxSQLNet across all difficulty levels. In the DBPal (Train) case, we see that with the addition of synthetic training data generated only using schema information from the training set, DBPal is already able to outperform the baseline SyntaxSQLNet model. This is due to the fact that our novel training pipeline is able to supplement the existing training data with additional query patterns (e.g., nested subqueries) that are not present (or numerous enough) in the training data. As shown, this helps significantly for the harder queries, with DBPal being able to outperform the baseline by more than $2\times$ for the "very hard" category due to the fact that the training pipeline introduces new query patterns (e.g., nested queries) to the model.

In general, DBPal (Full) is able to leverage additional query patterns from the synthetic data generation pipeline that are specific for the test schemas. With this information, DBPal (Full) is able to generate training examples that provide the model with additional information (e.g., table names, column names, column values) that is specific to test databases. As shown in Table 8.2, the added synthetic data for the test schemas in Spider when using DBPal (Full) is able to offer additional performance improvement over DBPal (Train). More concretely, with the help of the additional generated training data, we can further improve translation accuracy across all query difficulties of Spider by $15 - 27\%$.

| Algorithm | Naive | Syntactic | Lexical | Morph. | Semantic | Missing | Mixed | Overall |
|---|---|---|---|---|---|---|---|---|
| SyntaxSQLNet | 0.281 | 0.228 | 0.070 | 0.175 | 0.175 | 0.088 | 0.140 | 0.165 |
| DBPal (Train) | 0.930 | 0.333 | 0.404 | 0.667 | 0.228 | 0.088 | 0.193 | 0.409 |
| DBPal (Full) | 0.947 | 0.632 | 0.544 | 0.667 | 0.491 | 0.158 | 0.298 | 0.531 |

Table 8.3.: Patients Benchmark Results

### 8.6.2. New Benchmark: Patients

While Spider covers both a wide variety of schemas from different domains and different SQL query patterns, it does not comprehensively test different linguistic variations. Hence, we introduced a new open-source NL2SQL benchmark[1] that is available online specifically to test a model's linguistic robustness.

#### Setup

The schema of our new benchmark models a medical database comprised of hospital patients with attributes such as name, age, and disease. We refer to this dataset as the Patients benchmark. In total, the benchmark consists of 399 carefully crafted pairs of NL-SQL queries.

To better test the linguistic robustness of the given translation model, queries are grouped into one of the following categories depending on the linguistic variation that is used in the NL query: naive, syntactic paraphrases, morphological paraphrases, semantic paraphrases, and lexical paraphrases, as well as a category where queries have some missing information. These categories are formulated along the guidelines of paraphrase typologies discussed in [VMR11] and [BH13]. While the NL queries in the naive category represent a direct translation of their SQL counterpart, the other categories are more challenging: syntactic paraphrases emphasize structural variances, lexical paraphrases pose challenges such as synonymous words and phrases, semantic paraphrases use changes in lexicalization patterns that maintain the same semantic meaning, morphological paraphrases add affixes, apply stemming, etc., and the missing category includes implicit references to concepts.

Unlike other benchmarks that test for exact syntactic match of SQL queries, Patients tests instead for semantic equivalence. Since the test set is (relatively) small (i.e., 57 queries per category), we manually enumerated possible semantically equivalent SQL query answers. However, if the benchmark were to be extended, one could use an equivalence checker (e.g., Cosette [Chu+17]) to verify correctness.

In the following, we show an example query for each of these categories:

**Naive** *"What is the average length of stay of patients where age is 80?"*

**Syntactic** *"Where age is 80, what is the average length of stay of patients?"*

**Morphological** *"What is the averaged length of stay of patients where age equaled 80?"*

**Lexical** *"What is the mean length of stay of patients where age is 80 years?"*

[1] `https://link.tuda.systems/paraphrase-bench`

**Semantic** *"On average, how long do patients with an age of 80 stay?"*

**Missing Information** *"What is the average stay of patients who are 80?"*

#### Results

In this section, we show how our proposed techniques compare using the previously described Patients benchmark. Table 8.3 shows the performance of SyntaxSQLNet (Baseline), our proposed synthetic data generation using only information from the training set (DBPal (Train)), and synthetic data generation using schema information from both the training and testing set (DBPal (Full)).

In the results, we see that our proposed synthetic data generation techniques can help improve the performance of SyntaxSQLNet across all of the linguistic variation categories. In particular, our techniques improve the translation accuracy by almost $25\%$ by generating additional training data over only the training set and can provide a more than $35\%$ accuracy improvement over SyntaxSQLNet by leveraging schema information about the test databases.

In general, the results of our training data augmentation fall into two categories. On one hand, there are query pattern categories where the baseline DBPal augmentation achieves almost all of the observed performance improvement (e.g., Naive, Morphological). In these cases, DBPal improves model performance by providing training examples for classes of queries that are not well-covered by the Spider training set, and the target schema knowledge provides virtually no additional benefit.

The second category of query patterns is where there is a large performance difference between DBPal (Train) and the target schema augmentation version, DBPal (Full), where accuracy is often doubled (e.g., semantic, missing). In these categories, the additional schema information is particularly helpful because it allows the model to learn complex, domain-specific NL mappings. For example, consider the example semantic query: "On average, how long do patients older than 80 stay?" Clearly, the semantic meaning of the phrase "older than" refers implicitly to the "age" attribute of the patient, but this would not be easy to derive from a generic training set. However, by providing training data specifically generated from the target schema, DBPal is able to help the model to better learn these mappings.

### 8.6.3. Microbenchmarks

In the following, we present the results of our microbenchmarks, which include: (1) an analysis of Spider results based on SQL pattern coverage; (2) the sensitivity of DBPal when using only a fraction of seed templates; and (3) our hyperparameter optimization techniques described in Section 8.3.3.

| Algorithm | Both | DBPal | Spider | Unseen |
|---|---|---|---|---|
| SyntaxSQLNet | 0.375 | 0.000 | 0.244 | 0.013 |
| DBPal (Train) | 0.458 | 0.000 | 0.287 | 0.026 |
| DBPal (Full) | 0.462 | 0.250 | 0.317 | 0.040 |

Table 8.4.: Pattern Coverage Breakdown for Spider

#### Pattern Coverage Breakdown

To understand the specific benefits of DBPal, we analyzed our results for the Spider benchmark from Section 8.6.1 based on query pattern coverage in the training data. Table 8.4 shows the same overall performance results reported in Table 8.2 broken down by query patterns in the test set using the following categories: the query pattern of the test query was found in (1) both the Spider training set and augmented data generated by DBPal; (2) only the augmented DBPal data; (3) only the training set of Spider; and (4) neither of them. For example, the simple `SELECT COUNT(*)` query pattern appears in both training sets, whereas only the Spider training dataset has coverage for multiple nested subqueries.

In general, we see that DBPal improves accuracy for all four categories, demonstrating that our data augmentation process can improve linguistic robustness irrespective of which training set contains individual query patterns. That is, DBPal's generated data actually helps the model to generalize and be more linguistically robust to patterns that are not explicitly covered in our seed templates. This effect can be seen for patterns that appear only in the Spider training dataset (*Spider*), where DBPal improves the model performance by about 30%. Even more impressive is the over 3$\times$ improvement for test queries where the query patterns never explicitly appear in any training set (*Unseen*).

Again, as observed in our other results, the additional augmentation step using the target schema further increases accuracy. For the *Both* category, this enables model specialization of those patterns to the target schema, whereas for the *Spider* and *Unseen* categories, it helps the model to learn to translate patterns with no DBPal coverage to the target schema.

Finally, one notable result is for the query patterns that appear only in DBPal's seed templates. As expected, the baseline SyntaxSQLNet model has never seen these query patterns (since they do not appear in Spider) and thus has 0% accuracy, whereas DBPal achieves 25% accuracy by learning from augmented examples of these patterns.

#### Seed Templates

Since DBPal generates additional training data by instantiating seed templates, the number of templates used during training can impact the overall benefit of our training pipeline. Therefore, to demonstrate the impact of the seed templates, Figure 8.3 shows the normalized accuracy (i.e., performance compared to using all of the templates) using the Patients benchmark when varying the number of templates used during training.

For this experiment, we train the same SyntaxSQLNet model using the previously mentioned Spider training data and include additional training data that is generated for the Patients schema only using a random subset of the available seed templates. For example, in the 10% case, we augment

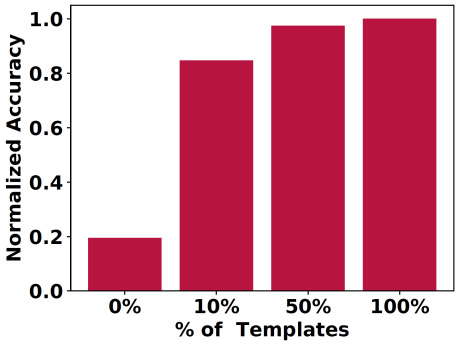



Figure 8.3.: Normalized Accuracy for Fractions of Seed Templates

the Spider data with additional training examples that are instantiated using a randomly selected 10% of the available seed templates on the Patients schema. Importantly, the random subsets are selected prior to instantiation, which means templates covering certain patterns are excluded.

As shown, the addition of only 10% of the available seed templates is able to improve the overall accuracy when running the Patients test queries by more than $4\times$. Adding even more of the available seed templates (i.e., 50%) offers an additional 15% accuracy improvement, showing that additional templates are able to capture distinct NL2SQL patterns.

#### Hyperparameter Optimization

As described in Section 8.3.3, we apply an automatic hyperparameter optimization procedure to tune the parameters of our training data generator. In this experiment, we show the results of applying our optimization procedure for generating the training data for the Spider benchmark we used in our experiments in Section 8.6.1.

As a test workload $T$ to tune the hyperparameters of our data generation pipeline, we used the full GeoQuery query test set of $280$ pairs provided by [Iye+17]. The rationale is that the GeoQuery queries are partially included in the Spider test set and thus represents a good test set for the hyperparameter tuning, since the queries can be seen as representative on the one hand but also independent of the actual Spider test set. For the experiment, we sampled $68$ random sets of hyperparameters. For every set of randomly sampled hyperparameters, we then trained a given model for up to a $6$ hour time limit (which we saw is the typical time a model needs to converge when trained on Spider and DBPal training data).

Figure 8.4 shows the distribution of the accuracy recorded from running the optimization procedure, which trains a model on every dataset that was generated using the randomly sampled hyperparameters. Of the $68$ parameter sets we evaluated, $59$ converged within their $6$ hour time limit. The worst model returned had an accuracy of $37.5\%$, while the best had an accuracy of $55.5\%$. The mean accuracy of all $59$ models was $48.4\%$, with a standard deviation of $3.5\%$.

We used the hyperparameters which returned the highest accuracy as the basis for all other experiments previously described in this section.

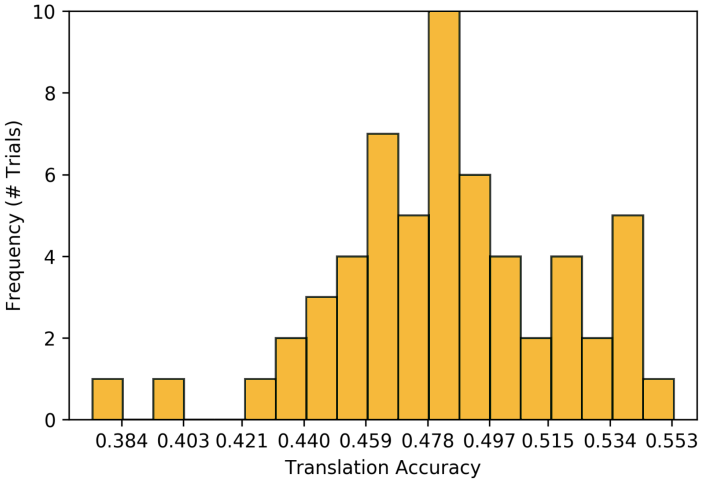



Figure 8.4.: Histogram of Test Accuracy for Enumerated Parameter Configurations

## 8.7. Related Work

The task of synthesizing SQL queries from natural language (NL) has been studied extensively within both the NLP and database research communities since the late 1970s [Pop+04; ZM96]. A 1995 study [And+95] extensively discusses challenges that need to be addressed pertaining to Natural Language Interfaces for Databases (NLIDBs); their list includes linguistic coverage and database portability.

NLIDBs have a long history in the database research community [And+95; LJ14a; LJ14b; PEK03; Ran+05b; Sah+16]. Most of this work relied on classical techniques for semantic parsing and used rule-based approaches for the translation into SQL. However, these approaches have commonly shown poor flexibility for the users who phrase questions with different linguistic styles using paraphrases and thus failed to support realistic scenarios.

More recent approaches tackled some of the limitations of the original NLIDBs. For example, the system ATHENA [Sah+16] relies on a manually crafted ontology that is used to make query translation more robust by taking different ways of phrasing a query into account. Yet, since ontologies are domain-specific, they need to be hand-crafted for every new database schema. On the other hand, the NaLIR[LJ14a] system relies on an off-the-shelf dependency parser that could also be built on top of a deep model. However, it still implements a rule-based system that struggles with variations in vocabulary and syntax. Our system attempts to solve both of those issues by being domain-independent as well as robust to linguistic variations.

Within the NLP community, this task is most commonly treated as a semantic parsing problem where the goal is to model a mapping of NL to a corresponding logical form, in this case SQL. Earlier works, such as [BEM14; BL14; LJK11; ZC07], employ variants of CCG parsers [CC04] to parse NL utterances into an executable lambda calculus notation. It should be noted that the grammar of logical form notation is far more simplistic than that of a complex query language like SQL; as such, a single NL query can be mapped to an arbitrarily complex SQL statement crossing many tables and involving many layers of nesting.

Recent success in employing neural network sequence-to-sequence (seq2seq) modeling for syntactic constituency parsing by [Vin+15] has spurred efforts in adapting the same solution for semantic parsing. That is, they pose logical form synthesis as a neural machine translation task, adapting systems for translating English to Czech or French to instead treat the logical form as the target foreign language. In both settings, mapping to lambda calculus [DL16; DL18] or directly to SQL [Cai+18; Fin+18; Iye+17], the seq2seq architecture has shown competitive performance with statistical approaches that rely heavily upon hand-crafted lexical features.

In general, seq2seq models consist of a large number of parameters that require vast amounts of training examples. This poses a substantial challenge, as collecting diverse enough training data comprising pairs of NL utterances and logical form or SQL queries requires expensive expert supervision. Iyer et al. [Iye+17] attempts to deal with this data bottleneck by performing an online learning mechanism in which the model alternates between training and making predictions. Human judges identify incorrect predictions that need to be corrected by a crowdsourced worker with SQL expertise.

Alternatively, a solution more similar to ours is introduced by [Wan+15], whose approach produces pairs of canonical utterances aligned with their corresponding logical forms using a seed lexicon. However, they again use crowdsourcing to paraphrase the canonical utterances into more fluent sentences that include syntactic alterations and context specific predicates. While less efficient than an on-the-fly system, this form of crowdsourced annotation is much less costly, given worker's SQL expertise is not required.

The main contribution of this work addresses the training data bottleneck from a slightly different angle. We attempt to completely eliminate any manual annotation effort by a user who is not well-versed in SQL. Rather, the user needs to be familiar only with the given new domain in order to sufficiently annotate the new schema's elements with their NL utterances. We argue that our extensive linguistically-aware templates provide a comparable breadth of coverage as that of manually collected training data. Additionally, our strategy of employing PPDB [PC16] to automatically paraphrase the sentence can approximate a human doing the same task.

Previous work on Natural Language Processing (NLP) has heavily relied on classical statistical models to implement tasks such as semantic parsing that aim to map a natural language utterance to an unambiguous and executable logical form [ZM96]. More recent results on semantic parsing such as [DL16; JL16] have started to employ deep recurrent neural networks (RNNs), particularly seq2seq architectures, to replace traditional statistical models. RNNs have shown promising results and outperform the classical approaches for semantic parsing, since they make only few domain-specific assumptions and thus require only minimal feature engineering.

An important research area aiming to allow non-experts to specify ad-hoc queries over relational databases are keyword search interfaces [YQC10]. Recently, there have been extensions to keyword-based search interfaces to interpret the query intent behind the keywords in the view of more complex query semantics [Ber+13; Blu+12; TL08]. In particular, some of them support aggregation functions, boolean predicates, etc.

Some recent approaches leverage deep models for end-to-end translation similar to our system (e.g., [Iye+17]). However, a main difference of our system to [Iye+17] is that their approach requires manually handcrafting a training set for each novel schema/domain that consist of pairs of NL and SQL queries. In contrast, our approach does not require a hand-crafted training set.

Instead, inspired by [Wan+15], our system generates a synthetic training set that requires only minimal annotations to the database schema.

Another recent paper that also uses a deep model to translate NL to SQL is [XLS17]. First, the approach in this paper is a more classical approach based on identifying the query intent and then filling particular slots of a query. In their current version [XLS17], they can only handle a much more limited set of NL queries compared to DBPal. Furthermore, their approach leverages reinforcement learning to learn from user feedback in case the query could not be translated correctly, which is an orthogonal issue that could also be applied to DBPal.

Finally, in addition to its primary focus on generating labels for unlabeled training data, Snorkel [Rat+17a] also incorporates data augmentation techniques to generate additional heuristically modified training examples [Dao+19; Rat+17b]. Unlike Snorkel, DBPal presents many optimizations that are specific to the task of NL2SQL translation, including slot-fill dictionaries, random word-dropout, and paraphrasing techniques to increase the linguistic robustness of the generated training data. Additionally, DBPal includes an optimization procedure for hyperparameter tuning that leverages schema information to further specialize the generated training examples for the target use case.

## 8.8. Conclusion & Future Work

In this paper, we presented DBPal, a fully pluggable natural language to SQL (NL2SQL) training pipeline that generates synthetic training data to improve both the accuracy and robustness to linguistic variation of existing deep learning models. In combination with our presented data augmentation techniques, which help improve the translational robustness of the underlying models, DBPal is able to improve the accuracy of state-of-the-art deep learning models by up to almost 40%.

Longer term, we believe that an exciting opportunity exists to expand DBPal's techniques to tackle broader data science use cases, ultimately allowing domain experts to interactively explore large datasets using only natural language [JPP17]. In contrast to the typical notion of one-shot SQL queries currently taken by DBPal, data science is an iterative, session-driven process where a user repeatedly modifies a query or machine learning model after examining intermediate results until finally arriving at some desired insight, which will therefore necessitate a more conversational interface. These extensions would require the development of new techniques for providing progressive results [Tur+18; Zgr+17] by extending past work on traditional SQL-style queries [Cro+16; Gal+17] and machine learning models [Sha+19].

Finally, we believe there are also interesting opportunities related to different data models (e.g., time series [Eic+17]) and new user interfaces (e.g., query-by-voice [Lyo+16]).

# 9. Netted?! How to Improve the Usefulness of Spider & Co. (DESIRES'21)


## Abstract

Natural language interfaces for databases (NLIDBs) are an intuitive way to access and explore structured data. That makes challenges like Spider (Yale's semantic parsing and text-to-SQL challenge) valuable, as they produce a series of approaches for NL-to-SQL-translation. However, the resulting contributions leave something to be desired. In this paper, we analyze the usefulness of those submissions to the leaderboard for future research. We also present a prototypical implementation called *UniverSQL* that makes these approaches easier to use in information access systems. We hope that this lowered barrier encourages (future) participants of these challenges to add support for actual usage of their submissions. Finally, we discuss what could be done to improve future benchmarks and shared tasks for (not only) NLIDBs.


## Bibliographical Information


The content of this chapter was previously published in the peer-reviewed work "Benjamin Hättasch, Nadja Geisler, and Carsten Binnig. 'Netted?! How to Improve the Usefulness of Spider & Co'. In: *Proceedings of the Second International Conference on Design of Experimental Search & Information REtrieval Systems, Padova, Italy, September 15-18, 2021*. Volume 2950. CEUR Workshop Proceedings. CEUR-WS.org, 2021. URL: `https://ceur-ws.org/Vol-2950/paper-08.pdf`". The contributions of the author of this dissertation are summarized in Section 3.2.

## 9.1. Introduction

In a world where ever more data is generated, processed, and relied upon, it becomes continually more significant that data is not only accessible to a small group of people. Information can be contained in text, relational databases, knowledge graphs, and many other formats—but users do not want to deal with heterogeneous sources. What they are interested in is accessing information easily. The borders between structured and unstructured information keep blurring: when using Google for factual questions, infoboxes might show the answer without the need to open a search result. That result might even be wrapped in a generated sentence when voice search was used, and nobody cares whether the sentence was extracted from a web page or generated from a database.

On the other hand, there are good reasons why these different ways of storing information exist. Information access methods should leverage the possibilities of each while providing convenient and ideally unified interfaces. With this goal in mind, natural language interfaces emerged as a data retrieval method, leveraging one of our most flexible and intuitive means of communication.

Relational databases are an essential type of information storage. To query them, users require knowledge of the domain, query language (e.g., SQL), and database schema. Contrarily, the vision for natural language interfaces to databases (NLIDBs) encompasses the ability of any user to interactively explore large datasets without help or extensive manual preparation work [JPP17]. As one of the biggest challenges, the application of NLIDBs requires the means to translate natural language (NL) into SQL queries (NL2SQL) (for a recent comprehensive overview of methods and open problems refer to Kim et al. [Kim+20]). However, before such NLIDBs can be used as one of many interfaces for information access (i.e., users can enter their information request using arbitrary words and get a correct answer without knowledge about the database), further research is needed.

**Contributions:** We show that current benchmarks, especially the Spider challenge [Yu+18b] and the related challenges SparC [Yu+19b] and CoSQL [Yu+19a] are not sufficient to measure all relevant aspects and support the emergence of ready-to-use NLIDBs. Yet, to foster research not only on NLIDBs but on systems that integrate and use them, we publish an API called *UniverSQL*[1] to integrate submissions to the challenges into research prototypes and existing systems. Its core functionality is a wrapper implementation to allow the execution of arbitrary queries on pre- or custom-trained models. We additionally provide two sample implementations of this wrapper for existing NL2SQL translators (EditSQL [Zha+19] and IRNet [Guo+19]). The code is published under an open source license.

Finally, we provide an overview of the advantages and flaws of Spider and other benchmarks and provide ideas on how the evaluation of NLIDBs could advance.

We hope that this research encourages the use of NLIDBs and further development of approaches and benchmarks. Hopefully, this will help make more information accessible to everyone, regardless of their of background.

[1] `https://link.tuda.systems/univerSQL`

**Outline:** The rest of the paper is organized as follows: After briefly describing the Spider challenge and its siblings in Section 9.2, we analyze how reproducible and usable the submissions to the shared tasks are in Section 9.3. In Section 9.4, we present our prototypical implementation *UniverSQL* that makes more of these systems usable for research. We examine strengths, weaknesses, and possible further developments of benchmarks in Section 9.5, before providing a brief final summary in Section 9.6.

## 9.2. What are SPIDER, SparC and CoSQL?

The Spider challenge [Yu+18b] has become one of the standard evaluations for NLIDBs since its publication in 2018. So far, it was cited 217 times and 71 submissions were made to the shared task. The dataset aims to surpass most existing datasets in size by at least one order of magnitude. At the same time it covers a diverse set of simple and complex SQL queries. This provides the necessary basis for data-driven systems to translate joins, nestings etc., and challenges them to do so to achieve good performance on the development and test data splits.

Alongside the dataset, Spider provides a shared task: Since such a dataset is expensive to create, it is not feasible to create one every time the NLIDB is applied to a new database. The authors suggest that this problem is solved by NLIDB systems capable of generalizing to new databases and performing well across domains. This idea is not entirely new: Systems by e.g., Rangel et al. [Ran+05a] or Wang, Berant, and Liang [WBL15] already attempted to be domain-independent in one way or another. However, Spider is the first dataset of its size, complexity and quality. The split ensures that each database occurs in exactly one set (training, development, and test). This provides a concrete task description and evaluation process, allowing accurate and comparable measurements of success.

Yu et al. [Yu+18b] also propose a way of categorizing SQL queries with regard to difficulty in the context of the translation task. The concept regards the number of SQL components, selections, and conditions to label a query as easy, medium, hard, or extra hard. A SQL query is estimated to be harder if it contains more SQL keywords, e.g., a query is considered to be hard if it contains nestings, the EXCEPT keyword, or three (or more) columns in the SELECT statements, three (or more) WHERE conditions, and a GROUP BY over two columns. Even more structures or keywords in one query are considered extra hard. The Spider shared task encourages the submission of models to show up in the leaderboard. There are two variants: the original task does not check value accuracy, but there is also a leaderboard for systems that handle/predict values (not just queries with placeholders).

SparC [Yu+19b] is the multi-turn variant of Spider. It deals with cross-domain semantic parsing in context and is comparable to Spider in size, complexity and databases. However, queries are arranged in user interactions, providing dialogue-like context. Therefore, it is not sufficient to just translate the current NL utterance into SQL, but information from previous queries has to be taken into account. Analogous to Spider, SparC features a leaderboard for variants with and without value handling.

CoSQL [Yu+19a] takes the challenge to the level of a real conversational agent. It consists of both dialogues and annotated SQL queries simulating real-world DB exploration scenarios. Therefore, the system has to maintain a state. CoSQL defines several challenges, the simplest one mainly

| | Spider (-v) | Spider (+v) | SparC | CoSQL |
|---|---|---|---|---|
| **Entries** | 62 | 7 | 17 | 10 |
| **Diff. appr.** | 51 | 5 | 15 | 8 |
| **- Publications** | 36 (58 %) | 6 (86 %) | 8 (47 %) | 9 (90 %) |
| **- Code** | 20 (32 %) | 4 (57 %) | 4 (24 %) | 5 (50 %) |

Table 9.1.: Analysis of the leaderboard entries for Spider (with (+v) and without (-v) value prediction), SparC & CoSQL. We checked how many different approaches are presented, how many of them reference a publication, and how often there is code to at least try to reproduce the approach.

| | Spider (-v) | Spider (+v) | SparC | CoSQL |
|---|---|---|---|---|
| **Repositories** | 15 | 2 | 4 | 5 |
| **- Empty?** | 2 (13 %) | 0 (0 %) | 0 (0 %) | 1 (20 %) |
| **- Code?** | 13 (87 %) | 2 (100 %) | 3 (75 %) | 4 (80 %) |
| **- Checkpoints** | 9 (60 %) | 2 (100 %) | 3 (75 %) | 2 (40 %) |
| **- Own data?** | 2 (13 %) | 0 (0 %) | 0 (0 %) | 0 (0 %) |

Table 9.2.: Analysis of the available repositories for the different challenges. We report whether the repositories are empty or contain code, whether checkpoints/pre-trained models are provided for download and whether the usage of this approach on own data/tables is in some way prepared.

adds further context to interpret compared to SparC, the other ones cover generation of suitable responses and intention detection/classification.

## 9.3. How Reproducible and Usable are the Challenge Submissions?

All three challenges (SPIDER, SparC and CoSQL) feature a public leaderboard where different approaches and their scores on the public, development as well as the unpublished test set are listed. In this section, we will investigate the state of the submissions, particularly with regard to how reproducible the submissions are and whether they can be used outside of the exact task. An overview of our analysis can be found in Tables 9.1 and 9.2 (as of June 2021). We will quickly interpret those results.

**SPIDER:** The leaderboard for the primary Spider task (without value handling) featured 62 entries in June 2021. Some of them are only small variations of the same system, nevertheless, this boils down to 51 different approaches. Yet, only little more than half (36 or 58 %) of those approaches are published in some way, the remaining approaches are anonymous or contain only names of authors or institutions (so far). For 25 submissions, a link to code is provided, yet, some repositories are empty or the link is invalid. In total, 20 approaches (32 %) have at least some code that could be used as starting point for reproduction. Unfortunately, this is not evenly spaced, only for two of the top ten current submissions (and for four of the top twenty) code is provided.

Two approaches deserve special mention: Shi et al. [Shi+20] provide a Jupyter Notebook for translation of user-specified queries on custom data[2] and the code of Lin, Socher, and Xiong [LSX20] allows interaction with pretrained checkpoints through a command line interface.

**SPIDER (with Value Predection):** This variation of the task (additionally covering value handling necessary for translating real NL queries) unfortunately received substantially fewer submissions (seven entries for five approaches and all but one with publication). Four approaches, provide code (only two of six publications).

**SparC:** Although this challenge was published just nine months after the Spider challenge, it received considerably fewer submissions so far. The leaderboard for the variant without value handling has 17 entries for 15 different approaches. For less than half of them (47 %) publications are referenced, for 24 % there is code and three submissions provide pre-trained models for download. For the variant with value prediction, there are only two entries, out of which only one references a publication and no code is provided at all. We therefore did not include this variant in Tables 9.1 and 9.2.

**CoSQL:** At the time of writing, the challenge was public for around 20 months. There were only baseline implementations or entries without publications for two of the three variants, only one entry included value handling. The main task received ten submissions by eight different approaches with a publication ratio of 90 %. For half of the approaches there is code, but in only two cases checkpoints can be downloaded and there is no preparation for the use of the models outside the evaluation scripts at all.

Overall, we have to conclude that reproducibility of the approaches submitted to the leaderboards of all challenges is at best mediocre, which is in line with problems of the community and especially research in computer science where reproducibility is still a challenge. ACM conferences try to tackle this through reproducibility challenges and badges in the ACM Digital Library.[3] Yet, publishing code and artifacts that allow others to redo the experiments is still optional.

While it is surely not feasible to change the whole publishing and reviewing process at once, we think that shared tasks are a good place to start. Of course, it is fine that submissions are anonymous until the approach was reviewed and published. But we advocate that once names are revealed, it should also be necessary to reference publication and code. Authors of a challenge set the requirements for submissions to be included in a leaderboard—and they should take advantage of that.

Moreover, it should be honored when authors of an approach or research prototype invest that extra time to make it directly usable for others and their research.

A very good example (from a slightly different domain) is SentenceBERT [RG19b]. Although it is an implementation accompanying a research paper, it is extremely easy to use: install via pip, import, specify which model to use. The installation scripts will install dependencies and the

[2] `https://github.com/awslabs/gap-text2sql/blob/main/rat-sql-gap/notebook.ipynb`
[3] `https://www.acm.org/publications/policies/artifact-review-and-badging-current`

system will download required files/checkpoints, making it possible to build research on top of it in minutes.

That case is already the cream of the crop, in many cases significantly less effort would help: pinning versions of dependencies (especially machine learning libraries often introduce breaking changes in just months), run the code on a second machine under a different username, add an installation script to download required external data or add environment variables for configuration. Each of these steps can make it substantially easier to run foreign code (or your own after a while). It is not about providing perfectly fast and robust industry-grade software for production use—that is something (academic) researchers usually cannot accomplish and also should not spend their time on—but to allow quick prototypical usage to decide whether it is feasible to use an approach in research and maybe investing time in improving it. We therefore argue that shared tasks like Spider should require this in the future for submissions to their leaderboards, and find it a great pity that most of the current submissions are difficult to reproduce and even more difficult to utilize for further research.

## 9.4. Does it Translate?

As shown in the last section, in June 2021 there were 86 submission in total for Spider and SparC. If one wants to build a system on top of them, currently one has to pick one of the best performing approaches from the leaderboard, obtain the code, install dependencies, download pre-trained models (if any) and then find a way to run the code not on the benchmark data but on individual natural language queries. There has to be a better way. The Spider and SparC challenges do not enforce a certain architecture (i.e., their aim is to foster research on all kinds of approaches to solve the task and not tie it down to, e.g., a hyperparameter optimization for a fixed architecture). This has the downside of making it even harder to use the resulting approaches in other applications. As a community service, we therefore provide a simple API implementation called *UniverSQL* that can be used in prototypes for information access, i.e., ones that use NLIDBs (and maybe other components) but do not focus on implementing them. The idea is that this API can be used as a unified interface to NLIDBs regardless of their architecture. This allows researchers to concentrate on their task—and allows them to make use of approaches that would otherwise be difficult to use.

*UniverSQL* is a small python application that serve as a translation server. The API allows unified access to most important functionalities (select a database, select a translator, do the actual translation) and some convenience and debugging functions like logging. It can be used for individual translations but also for (context preserving) multi-turn interactions as in the SparC challenge. An overview of available endpoints can be seen in Figure 9.1.

The core of *UniverSQL* is a wrapper implementation to allow running arbitrary queries on pre-trained models. We provide two sample implementations of this wrapper for systems from the Spider leaderboard: EditSQL [Zha+19] and IRNet [Guo+19]. It also includes a script to set up these two systems and download required dependencies and model dumps. We publish our code together with an extensive documentation how to create wrappers for other NL2SQL approaches and scripts for simple setup. We hope that this itself evolves into a challenge where researchers provide such a wrapper implementation and installation script for their approach and will therefore maintain a *ready to use* list as part of the published code.[4]

[4] `https://link.tuda.systems/univerSQL`

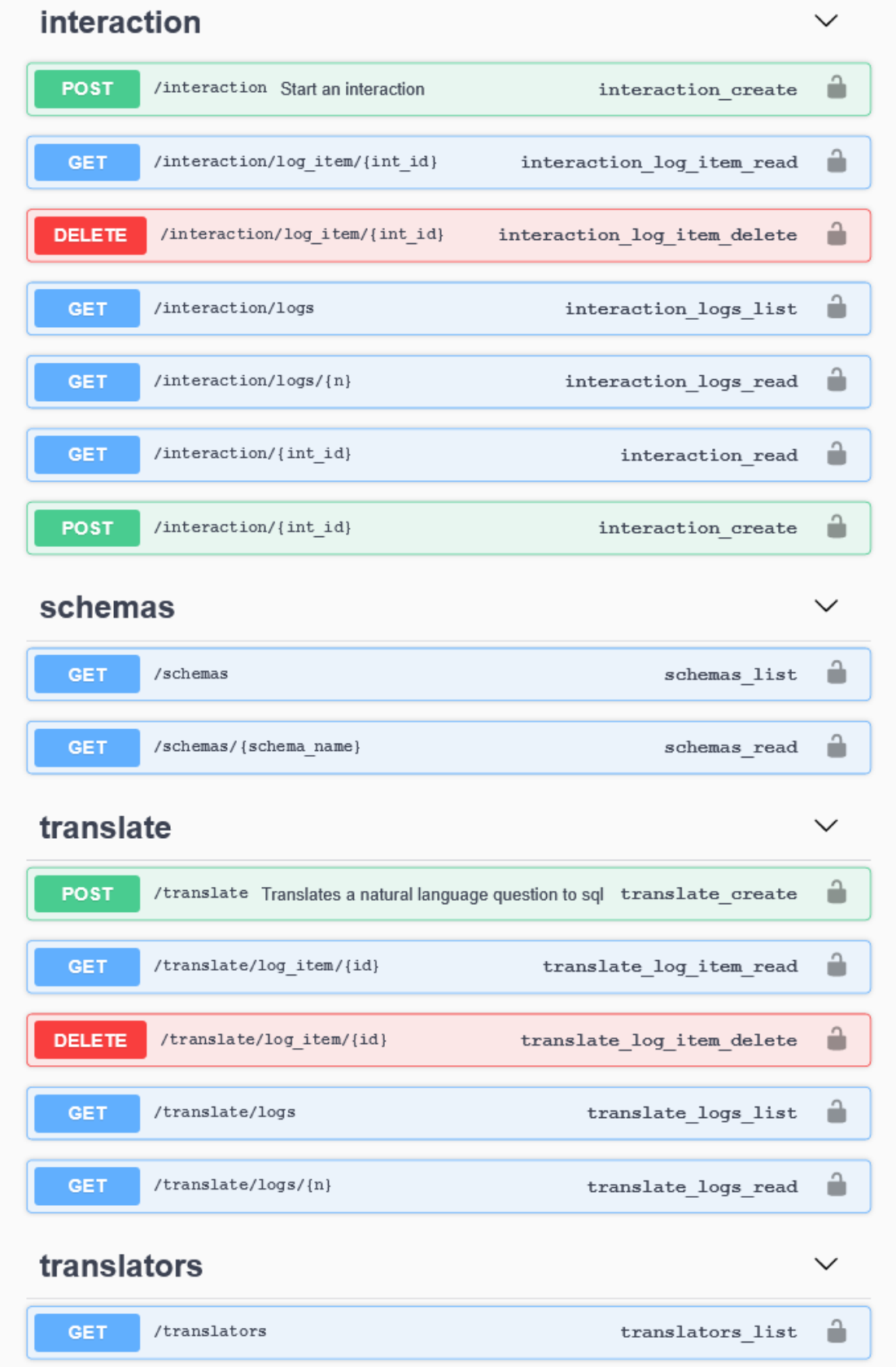



Figure 9.1.: Endpoints of the *UniverSQL* API

## 9.5. What Are We (still) Missing?

Modern data driven approaches would not be possible without big amounts of data, but curating and annotating it is out of scope for many researchers. Hence, it is not surprising that Spider and SparC, but also other datasets, have strongly advanced research in the field. However, we believe that further advancements are still possible:

We already outlined some flaws of Spider such as a missing focus on reproducibility. Yet, we also want to highlight advantages like the manually annotated and high-quality data, which deservedly currently makes it the most important benchmark for of NL-2-SQL translators.

In addition to Spider, there are other datasets and approaches for benchmarking of NLIDBs, but, like Spider, they have some flaws. We will take a glance at some of them, to outline typical problems:

The WikiSQL Benchmark by Zhong, Xiong, and Socher [ZXS17] is a large dataset (though smaller than Spider) that also features a leaderboard. Unfortunately, it consists only of a small number of unique query patterns [Fin+18] (in fact, half of the questions in the dataset are generated from one single pattern). In particular, it contains neither joins nor nestings. Furthermore, the NL questions are often low quality (i.e., many are grammatically incorrect), some do not have a proper semantic meaning and make little sense when read by humans and some NL questions do not have the same meaning as the associated SQL query.

Utama et al. [Uta+18] published *ParaphraseBench*, an approach that tries to measure translation difficulty by dividing queries into classes. The benchmark was manually curated but is quite small and covers only one table.

A recent paper by Gkini et al. [Gki+21] tries to benchmark existing translation systems. They focus on system aspects like execution times or resource consumption and not on translation accuracy. However, their analysis leaves some open questions: First, their dataset which is unfortunately not publicly available (yet) appears to be quite small, it consists only of 216 keyword-based and 241 natural language queries. Second, although they cite Spider, they did not include the high-performing approaches from the leaderboard in their evaluation. Overall, this approach does not appear sufficient for an evaluation that takes the user's perspective into account.

Even if we combined all these approaches, the result would still not be the best way to evaluate NLIDBs. Therefore, we will conclude with a brief outlook on what is still needed and what would be possible in this area.

As mentioned before, it would probably boost the usage of the approaches if they allowed for direct/easy use. Enforcing this is not an inherent part of a benchmark but could be done as part of the setup of a shared task.

Much more difficult but probably also even more important is taking the user's perspective into account. One way to do so could be end-to-end benchmarks that do not only evaluate the translation accuracy but the real performance in a data exploration task from input to the output (SparC and especially CoSQL do this to some extent). But there are many other highly interesting questions: We can measure the accuracy of a system like an NLIDB, but what accuracy should we strive for? Are all errors equally bad? Can a slightly wrong translation still be sufficient? What is the influence of a suboptimal translation? Will the user be satisfied by a system with $100\,\%$ translation accuracy? Or do they expect something that cannot be accomplished even by perfectly working systems? Answering such questions is hard, it can probably not always be automated, and it is difficult to

frame the answer as a bunch of numbers. Yet, a framework to assess a system in respect to these kinds of questions would help to better decide on which improvements it is worth to focus. We therefore hope that this user perspective will be considered more regularly in computer science research—not as a separate field of research but an integral part to drive research in a direction that is suitable to support humans best in whatever they want to accomplish.

## 9.6. Conclusion

In this paper, we analyzed the reproducibility and preparation for use in further research of the submissions to the Spider, SparC and CoSQL challenges. Unfortunately, we found that only for about 40 % of the submissions code is available and for even fewer submissions artifacts like pre-trained model dumps are provided. Additionally, the code is in most cases only capable to do the batch translation of specific data required for the evaluation scripts of the challenges, but not prepared for use on other real world data. We therefore presented a prototypical API implementation called *UniverSQL* that provides a simple interface for NL to SQL translation and boils down the task of adapting an approach for individual translation to implementing a simple wrapper class. The implementation is provided as open source software. Finally, we analyzed further shortcomings of Spider and other benchmarks and advocated for a stronger user perspective when designing similar benchmarks in the future.

# 10. Demonstrating CAT: Synthesizing Data-Aware Conversational Agents for Transactional Databases (VLDB'22)


## Abstract

Databases for OLTP are often the backbone for applications such as hotel room or cinema ticket booking applications. However, developing a conversational agent (i.e., a chatbot-like interface) to allow end-users to interact with an application using natural language requires both immense amounts of training data and NLP expertise. This motivates *CAT*, which can be used to easily create conversational agents for transactional databases. The main idea is that, for a given OLTP database, *CAT* uses weak supervision to synthesize the required training data to train a state-of-the-art conversational agent, allowing users to interact with the OLTP database. Furthermore, *CAT* provides an out-of-the-box integration of the resulting agent with the database. As a major difference to existing conversational agents, agents synthesized by *CAT* are data-aware. This means that the agent decides which information should be requested from the user based on the current data distributions in the database, which typically results in markedly more efficient dialogues compared with non-data-aware agents. We publish the code for *CAT* as open source.


## Bibliographical Information


The content of this chapter was previously published in the peer-reviewed work "Marius Gassen, Benjamin Hättasch, Benjamin Hilprecht, Nadja Geisler, Alexander Fraser, and Carsten Binnig. 'Demonstrating CAT: Synthesizing Data-Aware Conversational Agents for Transactional Databases'. In: *Proc. VLDB Endow.* 15.12 (2022). URL: https://www.vldb.org/pvldb/vol15/p3586-h%5C%e4ttasch.pdf". The contributions of the author of this dissertation are summarized in Section 3.3.

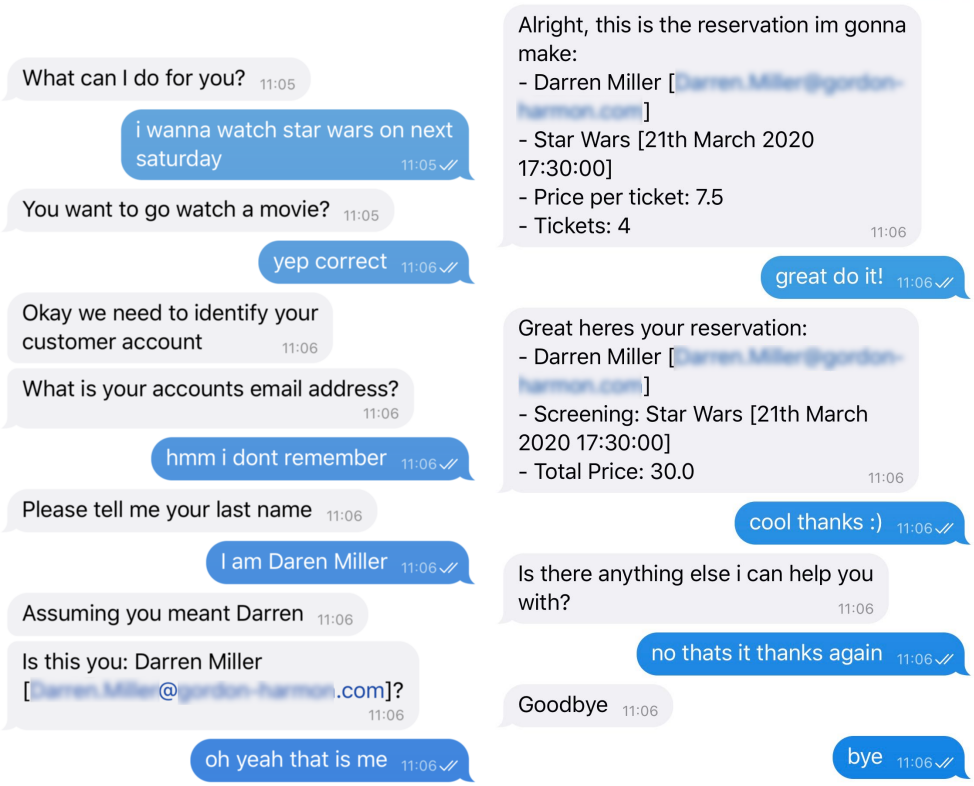



Figure 10.1.: Exemplary Dialogue with a Conversational Agent synthesized by *CAT*

## 10.1. Introduction

**Motivation:** Natural language interfaces are becoming ubiquitous because they provide an intuitive way to interact with applications such as web shops, online ticketing systems, etc. In particular, they allow users to directly express their needs instead of having to remember application-specific commands or the correct usage of user interfaces. Moreover, consumer products like Amazon Alexa or Apple Siri further raise the expectations of customers to interact using natural language. As a result, companies began developing conversational agents for supporting simple tasks or even basic business processes. For instance, a customer of an insurance company could report a claim or check the status of an existing report using such a conversational agent.

Yet, developing a task-oriented dialogue system for a given OLTP application (e.g., allowing users to buy a movie ticket) is a daunting task because this not only requires large amounts of annotated training data (i.e., actual dialogues between users and the system) for every application but also a manual integration with the existing database.

For instance, creating a conversational agent for a cinema ticketing system requires training data consisting of *user utterances* (e.g., “I want to reserve four seats tonight”), along with filled slots (e.g., `no_seats=4`) and annotated *user intents* (e.g., “reserve seats” or “inform about available shows”). These dialogues, however, are expensive to gather and annotating them is a large manual error-prone effort which requires extensive domain-knowledge. Worse, neither the training dialogues nor the integration with the existing database can be reused for a different domain.

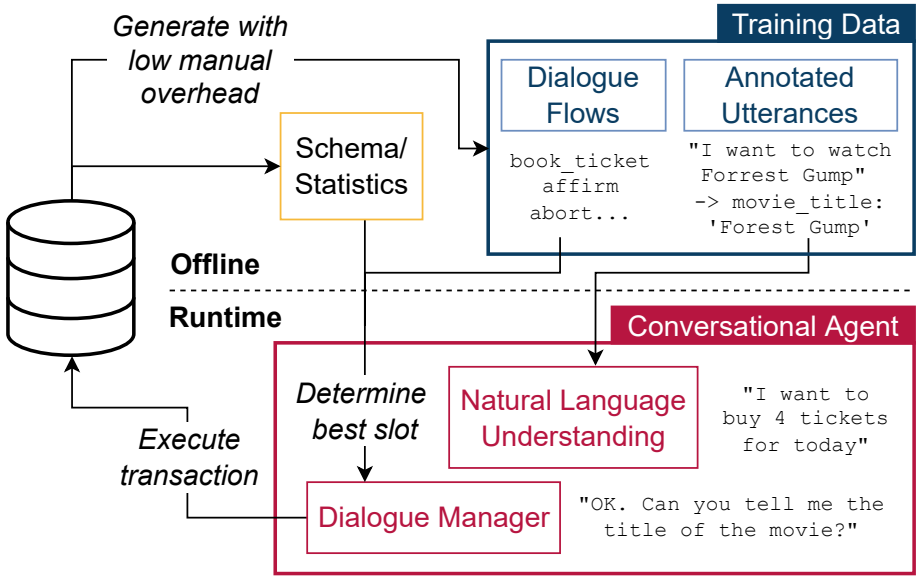



Figure 10.2.: Overview of *CAT*, showing both the creation and the usage of an agent.

Another drawback of existing approaches to build task-oriented dialogue systems is the lack of integration between the task-oriented dialogue system and the OLTP database, which is often the backbone of the business process. In current systems, a large amount of information must be provided manually even though it is already implicitly available in the database (for instance the required slots/attributes, the associated data types, the affected tables, etc.). Moreover, existing dialogue systems learn the order and types of information to request from the user purely from the manually created user dialogues. Not taking the data characteristics into account results in inefficient dialogues, as we describe below.

**Contributions:** In this demo we introduce *CAT*, a framework to synthesize conversational agents for a given database and a set of transactions (i.e., an OLTP workload with user-defined functions) with only minimal manual overhead. Given a database and a set of transactions, the user only has to provide a few example formulations for each intent instead of a large number of annotated example dialogues. Using a data-driven simulation, our approach generates annotated dialogues of possible user interactions from those intents, which can then be leveraged to train a conversational agent. This alleviates the extensive process of manually creating dialogues, which has to be repeated for every domain and database.

An inherent challenge is that for database transactions, it is often required to uniquely identify entities of the database. For instance, in order to book cinema tickets, the corresponding *customer ID* is required. Often the customer will not have the unique ID at hand but only information such as their name or address. In contrast to existing conversational approaches, *CAT* is data-aware; i.e., it considers the data characteristics at runtime to (1) deal with incomplete information (e.g., a customer who cannot remember an ID) and (2) request the most suitable information to narrow down the set of candidates as quickly as possible. Different from existing conversational approaches which take a pure learning-based approach to determine what to ask for, *CAT* uses information such as database statistics (e.g., selectivities). For example, once the user provided their name, the agent might ask them for the city they live in, knowing that based on the entries in the database

this is sufficient to uniquely identify the *customer ID* (while another name requires a different attribute to narrow down the options).

The contributions of this demo paper can be summarized as follows:

- **Automated Training Data Generation:** We suggest a procedure to automatically generate training dialogues given a database and a set of transactions with only minimal manual overhead. We then use it to train a conversational agent.
- **Data-driven Dialogue Policy:** We introduce a conversational agent policy that leverages the data characteristics to request information from the user to minimize the number of dialogue turns, i.e., to fulfill a user request as quickly as possible.
- **Demo Scenario:** We showcase *CAT* by a demonstration scenario with a fully synthesized conversational agent for a movie database which allows a user to reserve tickets, cancel existing reservations and list movie theater screenings. We show both the creation of the agent using our system and the usage of the agent.

**Outline:** In the remainder of this paper, we first introduce the system architecture of *CAT* (Section 10.2), before we define our training data generation (Section 10.3) and the data-driven dialogue policy (Section 10.4). Finally, we describe the demo application (Section 10.5).

## 10.2. Overview of *CAT*

The goal of *CAT* is to synthesize conversational natural language interfaces for database transactions while avoiding the shortcomings of existing task-oriented dialogue systems. To address these problems, *CAT* leverages the information about a given database and a set of transactions: this is done for training data generation with weak supervision, but also at runtime to take data characteristics into account to steer the user dialogue (e.g., the movie a user wants to see) more efficiently.

For instance, a cinema could have a customer database storing the reservations for movie screenings. A typical transaction to make accessible using a conversational agent is the ticket booking process, where the users have to specify their `customer_id`, the `screening_id` and the number of tickets. In order to integrate such a task into a typical existing task-oriented dialogue system, we would first have to model the tasks the conversational agent supports (e.g., buy a ticket) along with slots, i.e., the required attributes for the task (e.g., the `screening_id` and `customer_id`).

All this information, however, is typically already available in the given database and the set of its transactions (e.g., implemented as stored procedures or user-defined functions). Therefore, the main idea of *CAT* is to automatically extract and leverage this information instead of asking the user to manually specify it. Moreover, *CAT* then uses this information to synthesize annotated dialogues which are needed to train the conversational agent. Hence, instead of collecting this training data for every domain and database manually, we automate this process. Moreover, the agent and the database are tightly integrated afterwards, and the agent can directly execute the desired transactions without any manual overhead—in contrast to existing task-oriented dialogue systems where a dedicated database integration would have to be developed for every domain.

This tight integration also allows us to use characteristics of the given database (e.g., data-statistics) at runtime to guide the dialogue. For instance, to identify the movie a user is interested in, the agent asks the users for properties of the movie (e.g., genre or actors playing in the movie). In the following, we give a brief overview of how *CAT* works as depicted in Figure 10.2:

**Training Data Generation (Offline):** In order to generate a conversational agent, we require training data for both the natural language understanding (NLU) and the dialogue management (DM) models [ZE16]. The NLU model translates user utterances (e.g., "I want to watch 'Forrest Gump'") into annotated slots (`movie_title='Forrest Gump'`) and user intents (ticket reservation). For the NLU training, we generate utterances using a few base templates that are provided by the developer. To form full sentences from these templates, the existing data in the database can be used. In addition, we increase the variety of the natural language by using automated paraphrasing, as done by Weir et al. [Wei+19b] for natural language interfaces for databases. Furthermore, to learn typical dialogue flows, i.e., what high-level action to take next (e.g., retry a task after an abort), we generate additional training data using the idea of dialogue self-play [Sha+18], i.e., we simulate different users interacting with a conversational agent. *CAT* then uses this training data to train state-of-the-art models for NLU and DM using the RASA open source conversational AI framework.[1]

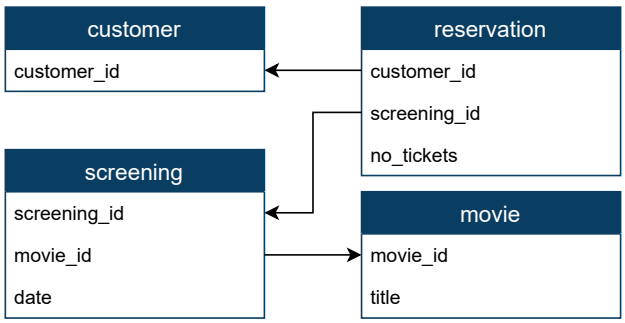
**Data-aware Dialogues (Runtime):** At runtime, the dialogue outlines created in the last step already determine the high-level flow of the dialogue. In addition, the conversational agent has to decide on the low-level flow, to determine which information should be requested next from a user to uniquely identify an entity required for a task, e.g., it could decide to ask for the movie title to identify the movie. In current approaches, this selection is usually done by learned models operating just on the previous input by this user [Sha+18]. In contrast, in order to efficiently narrow down the candidate movies, *CAT* takes information such as the selectivity of attributes already in the database into account. In addition, we allow adding an annotation to the database schema indicating which of the attributes are probably unknown to the customer. For instance, even though the `screening_id` is very useful and ultimately required for the transaction, the user will most likely not be aware of it and the conversational agent should thus not request it from the user. This results in more succinct dialogues, since the agent quickly gathers the information needed for a transaction.

In particular, the best information (i.e., a so-called slot) to request depends on (i) the probability that the user knows a certain attribute and (ii) how much this attribute narrows down the current set of candidates. Learning both factors end-to-end means learning the database content along with user preferences simultaneously, and again requires a large amount of data. We thus propose a different approach and explicitly keep track of the candidates (e.g., the screenings that match the previous user preferences) and request the next attribute based on the data distribution of the candidates and the likelihood that the user can provide this information. Moreover, while existing task-oriented dialogue systems implicitly assume that the database consists of just a single table [Sha+18], we can seamlessly integrate foreign-key dependencies, e.g., a user can provide information about actors to narrow down the set of possible screenings via the movie relation.

[1] `https://rasa.com/`

**Database and Transaction**

```
FUNCTION ticket_reservation(IN customer_id, IN screening_id, IN ticket_amount)
```




**Manually defined templates**

**Extracted Tasks and Schema Information**
`ticket_reservation`: `customer_id` (customer), `screening_id` (screening), `ticket_amount` (integer), …
**Natural Language Templates**
The movie title is {title}; I need {no_tickets} tickets; The screening is on the {date}; …

**Generated Training Data**

**DM Training Data**

```
customer: request_reservation
bot: identify_screening
customer: abort_task
```

**NLU Training Data**

”The movie title is Forrest Gump.” -> `intent: inform(movie_title);`
`slots: movie_title='Forrest Gump'`

Figure 10.3.: Exemplary inputs & results for *CAT*'s training data generation pipeline.

Another advantage of this data-awareness is that no retraining is required in case data changes. The updated database is simply leveraged at runtime to steer the dialogue.

## 10.3. Training Data Generation

Both the natural language understanding (NLU) and dialogue management (DM) [ZE16] components are learned models and thus require dedicated training data, which is expensive to collect. Consequently, we try to automate the training data generation as much as possible. We now describe the training data generation pipeline for both models, examples for inputs and results can be found in Figure 10.3:

**Dialogue Management (DM):** The high-level flow of dialogues in *CAT* is derived from training data synthesized using a so-called dialogue simulation [Sha+18]. *CAT* simulates typical dialogues between the conversational agent and the user who communicate with each other using predefined

actions (e.g., `request_reservation`). The set of possible actions in *CAT* is derived automatically from the transaction definition.

By sampling different user behavior during the simulation (e.g., sometimes performing the whole action and sometimes aborting it) the synthesized dialogue flows consist of different outlines that are later incorporated into the agent. Different from Shah et al. [Sha+18], we do not model the process of uniquely identifying entities in detail in this dialogue self-play, e.g., asking for the right slots to find the exact screening is not incorporated in this step. Instead, we only include the high-level action (e.g., `identify_screening`, see Figure 10.3). Which information from a set of candidates is requested to uniquely identify the screening is then decided at runtime (see Section 10.4).

**Natural Language Understanding (NLU):** Moreover, in addition to the training data for high-level dialogue flows, *CAT* also synthesizes training data for the NLU model. To this end, we require utterances of a user ('I want to see the movie 'Forrest Gump'") along with annotated slots (`title='Forrest Gump'`) and user intents (e.g., reserve a ticket or ask for information about a movie) as ground truth labels. Gathering this information is a substantial manual effort—collecting dialogues would come at the cost of simulating dialogues with testers. Even if dialogue traces are available, annotating them with the intents remains a manual effort. We thus take a different route, and let the developer specify a few natural language templates (e.g., "I want to watch {`movie_title`}"). By filling the placeholders with actual data stored in the database, we synthesize annotated natural language statements, which we automatically paraphrase afterwards to further augment the training data. Different from Shah et al. [Sha+18] where the user similarly specifies templates, we do not use crowdsourcing for this since this incurs high costs and might not be feasible for many transactions but instead utilize automated paraphrasing approaches.

**Initial Evaluation Results:** We compared several configurations of *CAT* to state-of-the-art approaches for intent classification and slot filling, using the widely used ATIS spoken conversation corpus [HGD90]. While all baselines require manually crafted training data, *CAT* only relies on synthesized training data, but still reaches comparable performance for slot filling. Moreover, on the intention classification task, *CAT* even outperforms multiple baselines.

## 10.4. Data-Aware Dialogues

We decide which information to request from the user for the unique identification of entities (e.g., ask for the movie title to find the screening) at runtime by keeping track of the current set of candidate entities (e.g., screenings that match with the already expressed user preferences) and select those attributes which narrow down this set as quickly as possible, the *informative* attributes. To do this, we choose the attribute with the highest entropy.

Note that the optimal attribute is not necessarily part of the table storing the entity. For instance, if a customer does not recall the exact movie title, it might be beneficial to ask for actors appearing in the movie. Since keeping track of candidates happens at runtime, it is not feasible to join every possible table with the set of candidates. Instead, we employ a priori information on the number of unique values of an attribute as well as the distribution of which attributes users were aware of

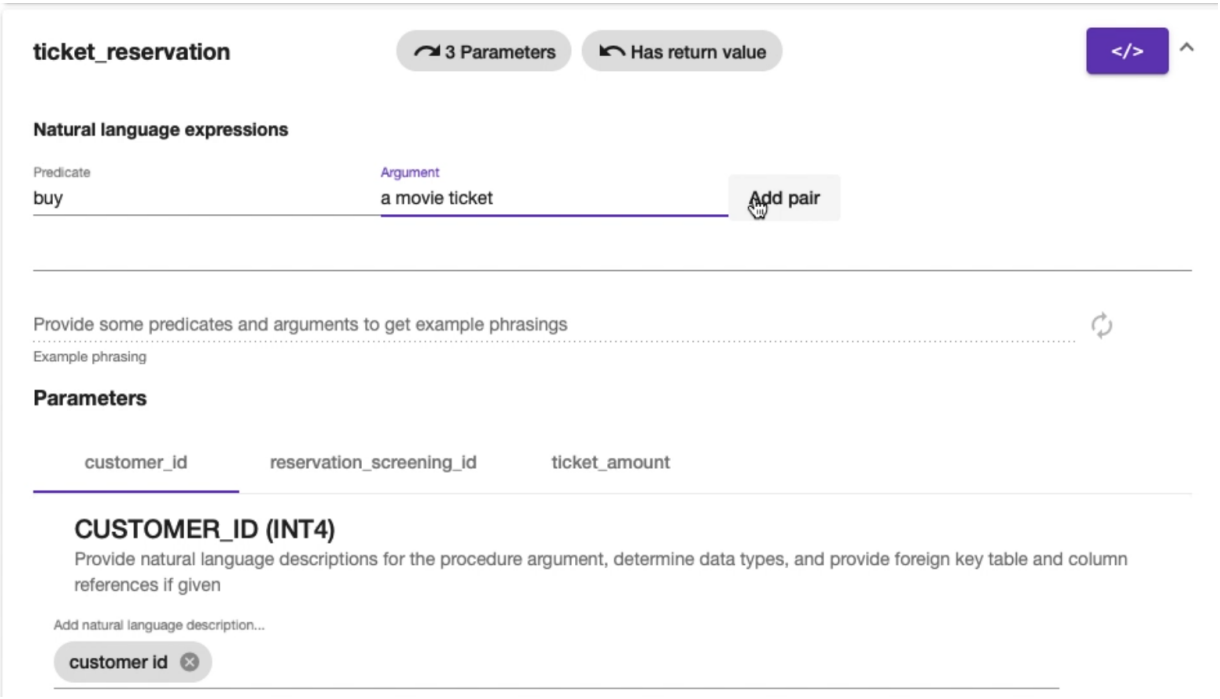



Figure 10.4.: Schema Annotation in *CAT*'s GUI

in previous sessions, and iteratively join additional tables to the current candidate set to provide improved next attributes to request from the user.

However, informative attributes are not useful if the user is not aware of them, e.g., while customer IDs quickly narrow down the set of customers, it is very unlikely that the user has such an ID at hand. Hence, the second dimension is the *User Awareness*. We address this two-fold: First, the developer can specify that certain attributes should preferably not be requested, e.g., IDs or other technical fields. Second, we learn from interactions with the conversational agent which attributes the users are likely to know. We combine both this probability and the informativeness of the attribute to score candidate attributes to request next.

**Initial Evaluation Results:** To evaluate the effectiveness of our data-aware selection policy, we compared it to static and random selection strategies using a movie database and again the ATIS dataset. The speedup (in terms of interaction turns) compared to a random strategy can be up to 80 % for large tables with many dimensions to join. When large amounts of data similar to the production entries are already available at training time, the static strategy can reach a similar performance as our data-aware policy, but will not adapt to data distribution changes at runtime. Additionally, it cannot react to systematic problems in uniquely identifying entries of some tables (caused by data characteristics like almost identical entries). An integrated caching strategy leads to an average response latency of only a few milliseconds.

## 10.5. Demonstration Scenario

In our demo, we showcase how a conversational agent for a cinema database supporting screening reservations and cancellations can be synthesized. It is fully integrated with the underlying database and allows users to interact using natural language to complete the domain-specific tasks.

To synthesize the required training data, we first annotate the schema and provide several natural language templates for the transactions using *CAT*'s GUI, as depicted in Figure 10.4. This is in fact the only database-dependent task for developers who want to synthesize an agent. We then start our training data generation to obtain both natural language statements for the NLU model and dialogue flows for the DM models. Afterwards, we trigger the training of these state-of-the-art models and generate the integration code with the database. With the completion of these steps, we have synthesized a conversational agent which interacts with users and triggers the right database transaction with the correct parameters at runtime.

The users can use this trained conversational agent to interact with the database as depicted in Figure 10.1. For instance, if the user wants to buy movie tickets, the agent will request the required information and execute the transaction upon confirmation. In the demo video, it can be seen how the agent identifies the intents and reacts to the user statements. It uses the information entered to identify their account, corrects misspellings, and asks the user to choose from a list of screenings fulfilling the preferences they have expressed. Finally, this triggers the execution of the transaction and the result is shown.

# 11. Towards Interactive Summarization of Large Document Collections (DESIRES'18)


## Abstract

We present a new system for custom summarizations of large text corpora at interactive speed. The task of producing textual summaries is an important step to understand collections of topic-related documents and has many real-world applications in journalism, medicine, and many more. Our system consists of a sampling component that ranks and selects sentences from a given corpus and uses an integer-linear program (ILP) to produce the summary. Both components are called multiple times to improve the quality of the summarization iteratively. The human is brought into the loop to gather feedback in every iteration about which aspects of the intermediate summaries satisfy their individual information needs. That way, our system can provide a similar quality level as an ILP-approach working on the full corpus but with a constant runtime independent of the corpus size.


## Bibliographical Information


The content of this chapter was previously published in the peer-reviewed work "Benjamin Hättasch. 'Towards Interactive Summarization of Large Document Collections'. In: *Proceedings of the First Biennial Conference on Design of Experimental Search & Information Retrieval Systems, Bertinoro, Italy, August 28-31, 2018*. Volume 2167. CEUR Workshop Proceedings. CEUR-WS.org, 2018. URL: `https://ceur-ws.org/Vol-2167/short6.pdf`". The contributions of the author of this dissertation are summarized in Section 4.1.




## 11.1. Introduction

Users like journalists or lawyers confronted with a large collection of unknown documents need to find the overall relation and event structure of those documents. An important step for this understanding process is to produce a concise textual summary that captures the information most relevant to a user's aims (e.g. degree of details or covered topics). While many automatic text summarization approaches have been suggested, there exist only a few that produce different summaries targeted at the individual user. One of those is the system recently proposed by

P.V.S. and Meyer [AM17].[1] A major limiting factor however is that their system does not scale for large corpus sizes since the runtime of their approach which uses an ILP solver at its core grows exponentially with the amount of sentences and may take hours for each iteration. This hinders the user from performing an adequate amount of feedback rounds to get a suitable level of quality and customization.

Therefore, in our work we build upon their system but introduce a ranking based sampling component. That results in a constant computation time of each iteration depending on the sample size instead of the corpus size. With this new approximate summarization model we can guarantee interactive speeds even for large text collections to keep the user engaged in the process. The original system consists of a web-based interface that allows the user to provide feedback and a backend which computes the summaries using an ILP. The user requests a summary and can then annotate the concepts of the summary i.e. mark them as important or unimportant for her current goal. This process will be repeated iteratively until the user is satisfied with the quality.

## 11.2. Overview

The main idea of this work is to enable the original system to achieve interactive response time on arbitrary large document collections with a similar quality of the resulting summary. A study [LH14] has shown that even small delays (more than $500\,\text{ms}$) significantly decrease a user's activity level, dataset coverage, and insight discovery rate, hence one should aim for lower runtimes.

Instead of looking at the complete document collection in every iteration, our approach only considers a sample from the documents per iteration and thus trades performance for quality of the summary. For creating the sample, two important factors thus play a role: The first factor is the *sample size* (i.e., the number of sentences in the sample), which determines the runtime of the summarization method; the second factor is the sampling procedure, that determines *which sentences* are part of the sample.

For deciding about the sample size, we need to be able to estimate the runtime for solving the ILP which mainly depends on its complexity (in number of constraints). In order to do so, we devised a cost function that maps the number of constraints to an estimated runtime. We use this function to derive the maximum sample size $k$ such that the runtime stays below a chosen interactivity threshold.

For deciding which sentences should be contained in the sample, we developed a novel heuristic called *information density* that is computed for each sentence. It ranks the sentences by the weight density of concepts in it normalized by the sentence length. We then only select the top-$k$ sentences based on this heuristic. The intuition is that sentences with a higher information density (containing more concepts rated as important) are more relevant to the user. With this sampling strategy, we are already able to archive a similar quality as the original system at a fraction of the runtime.

Our future work includes developing more advanced sampling strategies that can further improve the quality and increase the amount of feedback on different concepts. One direction would be to devise stratified sampling strategies using additional importance measures for (groups of) sentences. Furthermore, in addition to the current oracle-based approach for evaluation that gives feedback according to reference summaries we plan a user study.

[1] In order to get a better overview of how the system works, we recommend the readers to watch this video: `http://vimeo.com/257601765`

# 12. Interactive Summarization of Large Document Collections (HILDA'19)


## Abstract

We present a new system for custom summarizations of large text corpora at interactive speed. The task of producing textual summaries is an important step to understand large collections of topic-related documents and has many real-world applications in journalism, medicine, and many more. Key to our system is that the summarization model is refined by user feedback and called multiple times to improve the quality of the summaries iteratively. To that end, the human is brought into the loop to gather feedback in every iteration about which aspects of the intermediate summaries satisfy their individual information needs. Our system consists of a sampling component and a learned model to produce a textual summary. As we show in our evaluation, our system can provide a similar quality level as existing summarization models that are working on the full corpus and hence cannot provide interactive speeds.


## Bibliographical Information


The content of this chapter was previously published in the peer-reviewed work "Benjamin Hättasch, Christian M. Meyer, and Carsten Binnig. 'Interactive Summarization of Large Document Collections'. In: *Proceedings of the Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD 2019, Amsterdam, The Netherlands, July 5, 2019*. ACM, 2019. DOI: 10.1145/3328519.3329129". The contributions of the author of this dissertation are summarized in Section 4.1.

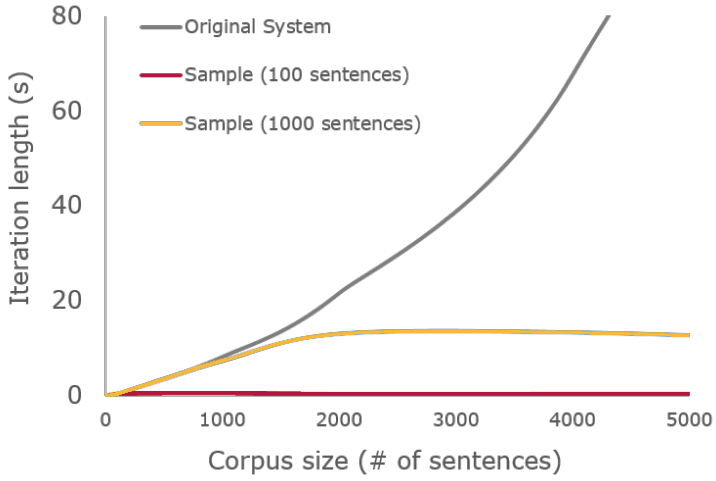



Figure 12.1.: Scalability of Text Summarization Models [Avi+18]

## 12.1. Introduction

**Motivation:** Existing data-centric systems for interactively manipulating, analyzing and exploring large data sets focus particularly on structured data. However, in many use cases the relevant data sources are not structured but are only present as a collection of texts. There already exist systems for text exploration like [Glo+13] and [FG17] that allow data scientists of varying skill levels and novice users to interactively analyze unstructured text document collections—however, those systems concentrate mainly on keyword searches and document ranking.

While keyword-based search systems are important to filter down the number of relevant documents, they still do not support users in semantically understanding the document collection. Imagine for example a journalist who just received a large collection of documents to start an investigative case, or a lawyer who needs to screen a large collection of e-mail conversations. In all these examples, an important step to better understand the collection of text and find the overall relation and event structure of those documents is to produce a concise textual summary that captures most of the important information relevant to a user's individual goal.

The task of producing textual summaries from a collection of documents is a well-established task in the text analysis community [NM11]. Despite a lot of research in this area, it is still a major challenge to automatically produce summaries that are on par with human-written ones. To a large extent, this is due to the complexity of the task: a good summary must include the most relevant information, omit redundancy and irrelevant information, satisfy a length constraint, and be cohesive. But an even bigger challenge is the high degree of subjectivity in the summarization task, as it can be seen in the small overlap of what is considered important by different users [AM17]. Optimizing a system towards one single best summary that fits all users, as it is assumed by current state-of-the-art systems, is highly impractical and diminishes the usefulness of a system for real-world use cases.

In a recent paper [AM17], we have shown that user feedback significantly improves the quality of the summary. However, each iteration of learning to create a new summary based on the user's feedback can take from several seconds for small document collections to hours for larger collections,

as shown in Figure 12.1 (black line). Since the customization of the summary depends on the user's feedback, it is one of the most important aspects to keep users involved in the exploration process. Yet this can hardly be reached with long iteration times of multiple hours, even waiting times of minutes or multiple seconds can already cause the user to lose interest. A previous study [LH14] has shown that even small delays of more than 500ms significantly decrease a user's activity level, dataset coverage, and insight discovery rate.

**Contributions:** In this paper, we present *Sherlock* that allows users to interactively summarize large text collections. In order to provide interactive response times in each iteration of the summarization procedure, we are using a novel approximate summarization model. The main idea of the approximate summarization model is similar to approximate query processing in databases: instead of looking at the complete document collection in every iteration, we only consider a sample from the documents per iteration to compute the summary. As a main contribution, we propose a method to select the sample size based on iteration time thresholds and evaluate multiple different sampling strategies. As we show in Figure 12.1, that way our approximate summarization model can provide interactive latency for each interaction loop independent of the size of the text collection that is being summarized (orange and blue line).

We already presented a demo of *Sherlock* at VLDB 2018 [Avi+18]. However, sampling on natural language is not a trivial task. In this paper, our main goal is studying the effectiveness of multiple sampling strategies and the impact of the sample size on the summarization quality. To this end, we employ importance-based and stratified sampling, and we benchmark them in a systematic experimental setup on different document collections.

A related paper was recently published for sampling training data for machine learning [Par+19]. In their paper, the authors suggest to use controlled sampling for architecture selection to predict the errors introduced by the approximate model. While their work concentrated mainly on classical models learned from structured data, we will investigate the effects of sampling for textual summarization models with different desired properties (e.g., subjective importance of concepts or wide vs. focused textual summaries).

**Outline:** The remainder of the paper is structured as follows: In Section 12.2 we describe the high-level idea of our system for interactive summarization and then propose different sampling strategies and explain the sample size estimation in Section 12.3. We then show an initial experimental evaluation of our novel sampling strategies on the model quality in Section 12.4.

## 12.2. System Overview

*Sherlock* [Avi+18], our system for interactive summarization, consists of two major components, as shown in Figure 12.2: a web-based user interface to collect the user's feedback and a backend that refines the text summarization model. The backend hosts multiple components: a document store (input docs in Figure 12.2) including the required indexes, the summarization component that accumulates the user feedback and learns to create summaries for every iteration as well as our approximate model to execute the summarization process at interactive speeds.

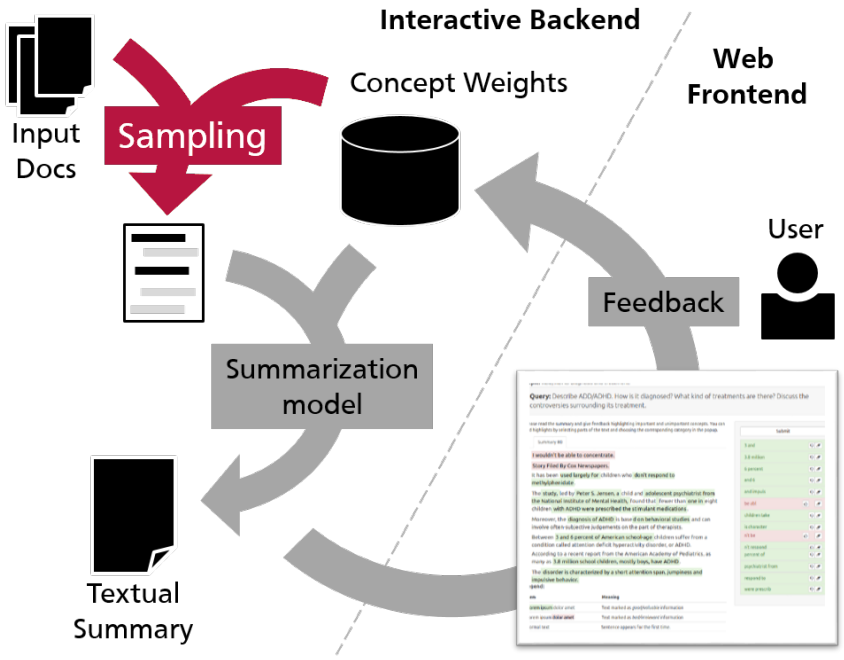



Figure 12.2.: System Overview of *Sherlock*.

**User Interface:** The web-based interface allows users to summarize a collection of textual documents in an interactive manner. A screenshot of the interface is included in Figure 12.2. In a typical setup, a user would need to read all the documents and manually summarize them. In our interactive setup, the user receives a summary, annotates all important and unimportant (parts of) sentences, and submits them as feedback for the next iteration where a refined summary is created by *Sherlock* and the user provides the next round of feedback.

**Interactive Backend:** The main task of the backend is to compute the summary for each iteration by taking the documents and the user feedback into account. In our system, we currently employ the summarization model as presented in [AM17] which maximizes the weighted occurrence of concepts in the summary by using an integer linear program (ILP). The first summary which is presented to the user is based on a model without any user feedback. Afterwards, in every iteration the summarization model is refined based on the user feedback of all previous iterations; i.e., the user can adjust the concept weights and hence the ILP needs to be re-executed. Instead of using the full document corpus as input, our backend uses samples of a given size (from very small to full data) resulting in different iteration times and expected quality. The details of our approximate summarization model are explained in the next section.

## 12.3. Approximate Summarization Model

The main idea of our approximate summarization model is to take the user feedback of the last iteration into account, adjust the summarization model, and then return a new version of the summary to the user. As discussed before, in order to achieve interactive response times in every

iteration, the approximate summarization model takes a sample of the overall document collection as input. The sampling strategy and the sample size have a big impact on the performance and the quality of the summarization model. In the following subsections, we discuss both these aspects in detail.

### 12.3.1. Sampling Strategies

In this first subsection, we discuss which sentences should be sampled in every iteration to refine the trained summarization model. There are numerous possibilities. In this paper, we suggest and evaluate three sampling strategies tailored towards training textual summarization models:

**Random:** This is the simplest sampling strategy where we just use a fixed number of randomly selected sentences in each iteration. This method is used as a baseline compared to the more sophisticated sampling strategies discussed next.

**Importance-based (called TOP-K as well):** Instead of sampling randomly, we suggest a second strategy that takes the importance of a sentence into account when sampling from the underlying document collection (i.e., more important sentences are sampled with a higher likelihood). Our intuition is that sentences with a higher *information density* (containing more concepts rated as important) are more relevant to the user. As concepts, we use bigrams in our system, as suggested by [AM17]. We initialize the weight of a concept using the *document frequency*; i.e., the number of documents in the collection the concept appears in. The information density of a sentence is the average weight of all concepts in the sentence. Based on the user feedback, we increase and decrease the weights of the concepts, yielding refined information density scores. In every iteration, we induce a sentence ranking and select only the top-$k$ sentences based on the information density. This strategy introduces some computation overhead but allows exploitation of collected feedback already in the sampling process.

**Stratified:** In the third sampling strategy, we additionally divide the input sentences into a fixed number of clusters based on sentence embeddings [Con+17]. Each of those clusters is individually ranked as discussed before. Based on the feedback, more sentences from clusters with better feedback are sampled. This allows dealing with diverse topics and causes the sampling to initially better explore all information available (instead of only the frequent concepts).

### 12.3.2. Sample Size Estimation

For all the before-mentioned sampling strategies one needs to choose the sample size as a parameter. This parameter should not be selected arbitrarily since a too small or too large sample size might have a negative impact on the overall quality—by not sampling relevant sentences as well as by changing the runtime of the summarization procedure which is important to enable the user to give interactive feedback.

At the core of our system, as discussed before, an ILP solver is used that maximizes the accumulated weight of all concepts in the summary for a given summary length (i.e. consists of sentences mainly

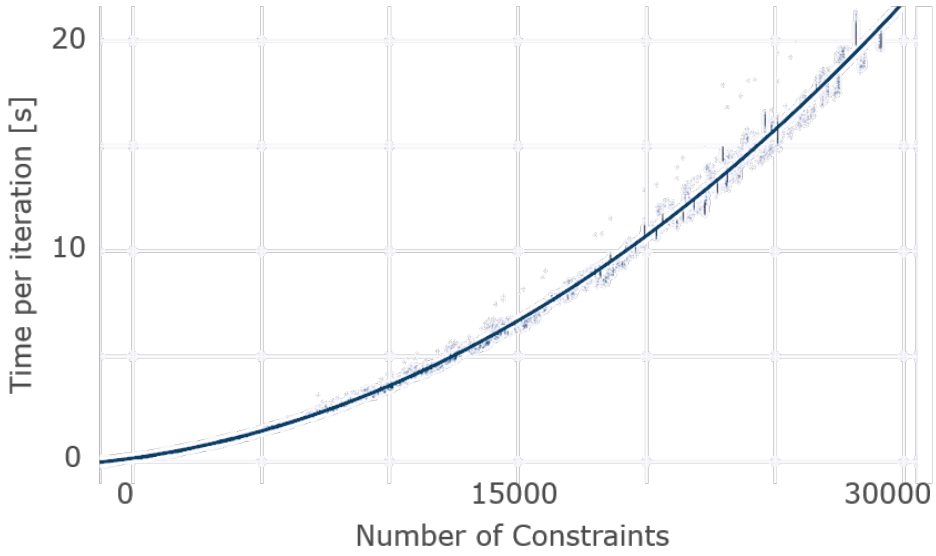



Figure 12.3.: Number of constraints in ILP in relation to estimated / actual runtime for finding the best summary (in seconds).

containing the highest-rated distinct concepts based on the user feedback). In order to set the sample size, we use a *cost model* to estimate the response time of the system. In future, we will extend this cost model to enable us to estimate the resulting quality of the summarization model when using only a sample of a given size.

The main intuition behind our cost model is that each sentence in the input to the ILP produces additional constraints that have to be respected for finding the summary in the next iteration by the solver. Fewer constraints make it easier for the solver to find a solution and therefore reduce the computation time. The cost function thus only depends on the sample size (which directly translates into the number of constraints) but not the summary length, since this is only present as a single constraint in the ILP. Hence, different summary lengths with the same amount of input concepts will still yield similar runtimes of the summarization system. We verify those findings at the beginning of our evaluation (Section 12.4.1).

## 12.4. Experimental Evaluation

### 12.4.1. Exp. 1: Sample Size

In a first experiment, we analyze the accuracy of our cost model to estimate the sample size. As discussed before, the cost function in *Sherlock* maps the number of constraints to an estimated runtime. We use this function to derive the maximum sample size $k$ such that the runtime stays below a chosen interactivity threshold (e.g., $500\,\text{ms}$). Figure 12.3 compares the actual runtime of the ILP model with the estimated runtime of our cost model (blue line). Both show a Pearson correlation coefficient of 0.96 to 0.97.

Furthermore, another important question is the effect of using a sample as input to the summarization model. Figure 12.4 shows the results of running the summarization procedure on different

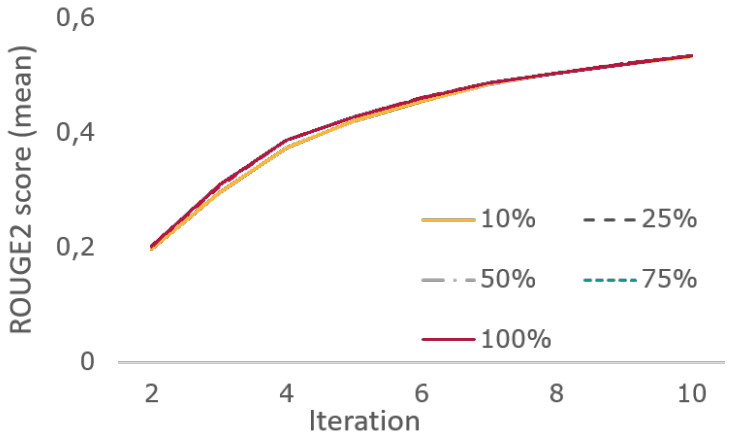



Figure 12.4.: Mean recall of bigrams (ROUGE2 metric) for running the system with 5 different sample sizes (from 10% to full data) on 615 different data sets (artificial and real data). All experiments use the TOP-K strategy.

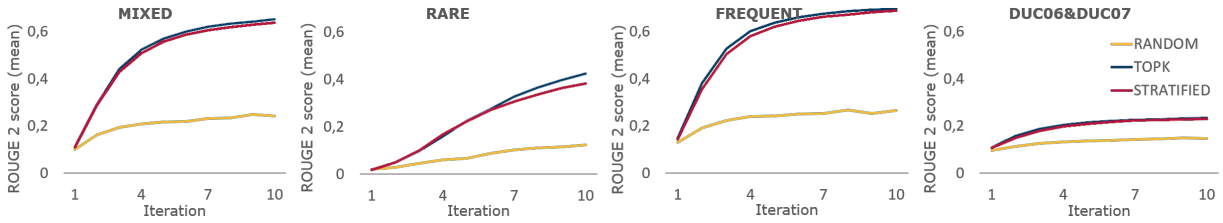



Figure 12.5.: Mean ROUGE2 scores for running the system for 10 iterations with three different strategies (RANDOM, TOP-K, STRATIFIED) on our summaries (MIXED, RARE and FREQUENT from left to right) as well as real data (rightmost plot, original DUC06 & DUC07). 380 different summaries were used per plot. ROUGE2 scores are always between 0 and 1, higher scores are better.

sample sizes using the TOP-K strategy (while STRATIFIED behaved similar in this experiment) over the different iterations (shown on the x-axis) to collect feedback from users. For collecting feedback, we simulate users in all our experiments who give perfect feedback to produce summaries (i.e., users always mark concepts as important if they appear in the gold summary). As a result, we see that the quality of summaries produced with only 10% sample size is nearly the same as for the full data and the mean quality of the system working on a quarter of the input data is indistinguishable from the mean quality of the original system while the length of each iteration is only about 20% of the one from the original system.

Another interesting observation is that the sample cannot be arbitrarily small—it still has to contain enough sentences to fill the full summary and the summarizing model should still be able to choose from different sentences. In our current prototype, we thus select the sample size based on our cost model such that the desired interaction threshold is met but the sample size still contains enough sentences to fill the full summary (i.e., sample size must be larger than the summary length).

### 12.4.2. Exp. 2: Sampling Strategy

In the second experiment, we evaluate the different sampling strategies and their robustness for different kinds of summaries. There are two important dimensions of textual summaries that have a major impact on how well a sampling strategy works: The first dimension is which concepts are included in a summary (i.e., frequent ones or rare ones). The second dimension is whether the summary is topically focused (for users who are interested in particular details) or if it contains a wide range of concepts (for users who want to get an initial overview).

In order to evaluate our sampling strategies on these different summary types, we create different summaries (called gold summaries) that follow the above-mentioned properties. To control the content of the summaries and make sure they still have the syntactic properties and word distributions of a real text, we use sentences from the existing summarization corpora DUC06 and DUC07 to create the gold summaries.[1]

Below, we first describe the setup for two experiments we conducted and then discuss the results at the end. We fix the sample size to 20% of the input documents for all different sampling strategies (RANDOM, TOP-K, STRATIFIED).

**Exp. 2a: Rare vs. Frequent:** In this experiment, we test the dependence of the sampling strategies on the frequency of the concepts. We therefore create three different classes of summaries: (1) summaries with random concepts containing both rare and frequent concepts (called MIXED), (2) summaries with concepts that appear frequently (called FREQUENT), and summaries with only concepts that are rare (called RARE). Furthermore, we use the summaries included in the original DUC corpora.

**Exp. 2b: Focused vs. Wide:** In this experiment, we want to test the sampling strategies on summaries with a rather focused or a wide set of topics. In order to create those summaries, we use the GloVe word embeddings [PSM14] and sample sentences where words are uniformly distributed over the vector space (called WIDE) or sentences where words are clustered in a certain region (called FOCUSED).

All gold summaries have a length of about 250 words.

**Discussion of Results:** The results of Exp. 2a (Rare vs. Frequent) are shown in Figure 12.5. Again, as before, we show the ROUGE2 metric over the different iterations of producing a summary, whereas users provide perfect feedback in each iteration. For the result in Figure 12.5, we see that TOP-K and STRATIFIED sampling clearly outperform the RANDOM sampling strategy and thus both can be used to generate high-quality summaries in only a few iterations. Moreover, as expected, both strategies work better on the summaries of types MIXED and FREQUENT and are less effective on summaries of type RARE since the sampling strategies prefer more important concepts over less important ones.

Our system also works reasonably well on the original summaries from the DUC06 and DUC07 corpora, but the scores are much lower (see right-most plot of Figure 12.5). Yet this is not caused by the algorithm or sampling strategies but by the fact that the gold summaries of the DUC corpora

[1] `https://www-nlpir.nist.gov/projects/duc/data.html`

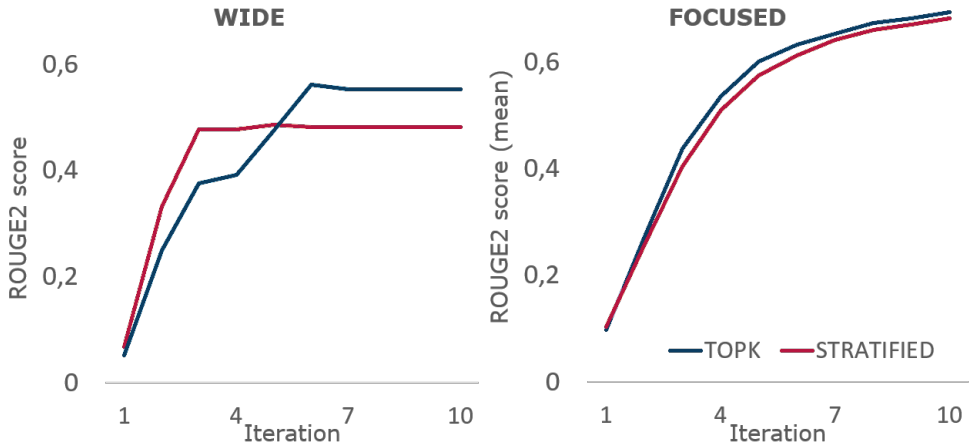



Figure 12.6.: ROUGE2 scores for running the system for 10 iterations on a WIDE summary (left) and mean ROUGE2 scores for 50 artificial datasets with FOCUSED summaries (right).

are abstractive and not extractive (i.e., summaries are not composed of full sentences of the input documents but handwritten by users). This is in contrast to the summaries used for the left three plots of Figure 12.5 which are extractive, leading to a higher upper bound for the ROUGE2 metric.

Finally, a clear difference between the TOP-K and STRATIFIED sampling strategy can be seen in the results for Exp. 2b (Focused vs. Wide) as shown in Figure 12.6. Here, the STRATIFIED strategy shows a clear benefit over TOP-K when used on a WIDE summary (left plot) instead of FOCUSED ones (right plot) at least in the first few iterations. The intuition is that STRATIFIED sampling is better able to include sentences required for the breadth of WIDE summaries. The reason why TOP-K performs better in the latter iterations is that the sampling is more efficient (and covers also the desired breadth) when enough user feedback was collected. In the future, we thus want to look into combinations of TOP-K and STRATIFIED sampling.

# 13. Summarization Beyond News: The Automatically Acquired Fandom Corpora (LREC'20)


## Abstract

Large state-of-the-art corpora for training neural networks to create abstractive summaries are mostly limited to the news genre, as it is expensive to acquire human-written summaries for other types of text at a large scale. In this paper, we present a novel automatic corpus construction approach to tackle this issue as well as three new large open-licensed summarization corpora based on our approach that can be used for training abstractive summarization models. Our constructed corpora contain fictional narratives, descriptive texts, and summaries about movies, television, and book series from different domains. All sources use a creative commons (CC) license, hence we can provide the corpora for download. In addition, we also provide a ready-to-use framework that implements our automatic construction approach to create custom corpora with desired parameters like the length of the target summary and the number of source documents from which to create the summary. The main idea behind our automatic construction approach is to use existing large text collections (e.g., thematic wikis) and automatically classify whether the texts can be used as (query-focused) multi-document summaries and align them with potential source texts. As a final contribution, we show the usefulness of our automatic construction approach by running state-of-the-art summarizers on the corpora and through a manual evaluation with human annotators.


## Bibliographical Information


The content of this chapter was previously published in the peer-reviewed work "Benjamin Hättasch, Nadja Geisler, Christian M. Meyer, and Carsten Binnig. 'Summarization Beyond News: The Automatically Acquired Fandom Corpora'. In: *Proceedings of The 12th Language Resources and Evaluation Conference, LREC 2020, Marseille, France, May 11-16, 2020*. European Language Resources Association, 2020. URL: https://aclanthology.org/2020.lrec-1.827/". The contributions of the author of this dissertation are summarized in Section 4.2.

## 13.1. Introduction

**Motivation:** Abstractive summaries help users to understand new text collections efficiently but writing these summaries is time-consuming and complex. Automatic summarization aims to eliminate or reduce the manual process. Since the advent of deep learning, automatic summarization methods require huge corpora to properly train neural architectures. The CNN/DailyMail dataset [Her+15], Gigaword [NGV12], and the New York Times (NYT) corpus [PXS18] are currently the largest summarization corpora. They have been used successfully for training a wide range of neural architectures, including recurrent neural networks [NZZ17], pointer-generator networks [SLM17], attention mechanisms [GDR18; PXS18], and approaches based on reinforcement learning [GMG18; NCL18].

All of these corpora are limited to the news genre where texts are typically too short to qualify as general-purpose summaries. For example, CNN/DailyMail provides only bullet-point summaries, Gigaword contains headlines as the summary of an article's first sentence, and the NYT corpus pairs news articles with their abstracts. To break new ground in the automatic summarization of other genres, we require new corpora that can cover other text genres and summary types on the one side but are large enough to train neural networks on the other side. However, constructing summarization corpora is still a manual task today and thus requires excessive resources which limits the variety of available corpora significantly.

**Contributions:** In this paper, we propose a novel approach to automatic construction of large summarization corpora. The main idea behind our approach encompasses the use of existing large text collections (e.g., thematic wikis) and automatically classifying whether the texts can be used as (query-focused) multi-document summaries as well as aligning them with potential source texts.

As an important first step for developing such an automatic construction approach, we use the Fandom wikis (formerly known as wikia). `Fandom.com` is a community page dedicated to providing a place for enthusiasts to discuss and share knowledge about their favorite entertainment content. It currently consists of more than 385,000 communities on different franchises (movies, television series, games, books, and more) with over 50 million pages in total. The sizes of the different communities range from only a few pages to well over 100,000 content pages. Most of those wikis use an open *Creative Commons Attribution Share-Alike license*, allowing us to use and redistribute their articles.

The Fandom wikis often contain articles describing the same topic in multiple levels of detail—there are articles giving a general overview of a character, event or place as well as articles focusing on a single aspect of it (e.g., a relationship, scene or time) in detail. Those articles normally reference each other through links. Our main idea is to automatically identify such overview articles or sections that qualify as a summary and align them with the potential source documents (i.e., the detailed articles) if the supposed alignment quality is high enough.

We show that it is possible to generate multiple different corpora with user-defined properties using this idea. For example, it is possible to vary the target length of the summaries, but also the difficulty of the summarization task which we control by the ratio between the sizes of summary and the source documents. Finally, we also allow users to choose whether the contents of a constructed corpus should be retrieved from a single community or whether a more general corpus is constructed from multiple communities at once.

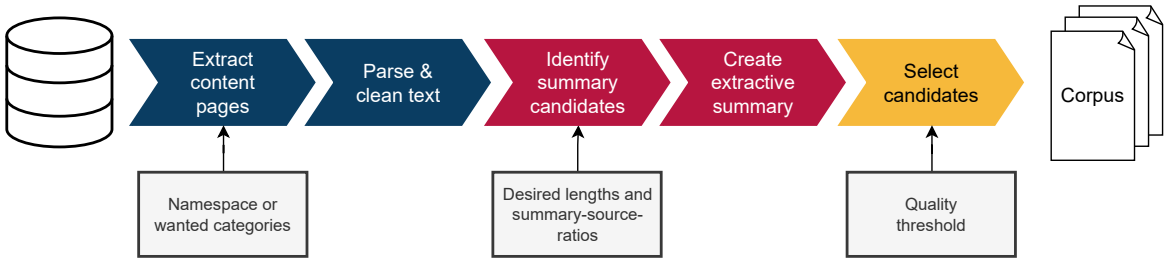



Figure 13.1.: Fandom Corpus Construction Pipeline

To summarize, in this paper we make the following contributions: (1) We present a framework that can be used to create new summarization corpora and discuss reasonable choices for the parameters. (2) We provide three new sample corpora created with our automatic construction pipeline. (3) We provide a comprehensive evaluation based on these corpora, using traditional and neural network based methods to validate that our pipeline is able to automatically create corpora of use for state-of-the-art summarizers. (4) We make our code available as open source under the MIT License. It can be found along with the data and a user documentation at `https://link.tuda.systems/fandom-corpora`.

**Outline:** We first give an overview of our automatic corpus construction pipeline and report important properties of our approach. Then we show the results of our evaluation before discussing potential future work and wrapping up our contribution.

## 13.2. Automatic Corpus Construction

In this section, we describe the steps of our automatic approach to create topic-specific multi-document summarization corpora. The essential stages of our approach are: (1) parsing and cleaning of input documents, (2) selecting potential candidates for abstractive summaries from those input documents and assigning summary candidates to them, and (3) choosing the final set of abstractive summaries based upon a newly developed quality threshold and splitting the selected summaries into training, validation, and test set if needed. An overview can be found in Figure 13.1.

As mentioned before, in this paper we use the Fandom wikis as an example set of source documents, but we believe that our approach can be easily extended to other sources: while step (1) is source specific and has to be implemented for each new set of sources, steps (2) and (3), which are the core of our automatic construction approach, are implemented in a general way.

### 13.2.1. Overview of the Pipeline

**Parsing and Cleaning the Sources:** The first step of our pipeline encompasses parsing the sources and cleaning the data for the automatic corpus construction.

As already mentioned, we use the Fandom wikis as a document source in this paper. Database dumps can be downloaded from the “Special:Statistics” page of each Fandom community. Each

| | |
|---|---|
| Min. summary length [words] | 150 |
| Max. summary length [words] | 400 |
| Extractive summary length [words] | 250 |
| Target-source-ratio | 2 |
| Min. source doc count | 5 |
| Min. bigram overlap | 50% |

Table 13.1.: Parameter settings for corpus creation used for the sample corpora of this paper.

dump consists of a large xml-file containing all contents of that wiki. In addition to the articles, this covers metadata on media files, discussion pages, category overviews, special pages and other sites not relevant to our task. For example, the English Star Wars wiki[1] contains about 150,000 content pages but over half a million pages in total. Hence, all non-article pages have to be discarded. This can be done by specifying an article namespace to use or by using rules to ignore certain page title prefixes (e.g., "Help:" or "Talk:").

Afterwards, the contents are preprocessed: We split the pages into sections including their respective titles, extract the links between pages and convert the content into plain text. This includes removing link texts, tables, templates and other kinds of wiki markup.

**Finding Summary Candidates:** The identification of summary candidates is the most crucial step for creating high-quality corpora automatically. At a high-level, a corpus that is useful for abstractive summarization should group a set of documents with at least one possible summary of these documents. In addition, many of the automatic summarization approaches take a "query" as input that represents the information needs of the user i.e., describes what aspects are important.

Hence, in this step, we aim to select triples (i.e., a set of source documents, a summary, and a query) that represent good candidates from a given cleaned data dump for the final corpus. For both the *source documents* and *summaries* our pipeline uses sections of the wiki articles since they are coherent and self-contained portions of text. As a *query* describing the section, we combine the title of an article with the section title, e.g., "Luke Skywalker: Birth".

To identify sections that qualify as possible summary candidate triples we use the following heuristics: (1) Only sections with a length between certain threshold values are considered as summaries. These thresholds can be adapted based on the task at hand. The default values for all parameters used for the sample corpora in this paper can be found in Table 13.1. (2) We discard summary candidates having only few linked documents (i.e., potential source documents). Again, the number of source documents is a parameter that can be set by the user. Higher values increase the difficulty of the summarization task since the summary content has to be extracted from more input documents, but may also drastically decrease the number of candidates overall. (3) After applying these purely statistical heuristics, we compute the content alignment between summary and source documents as the overlap between sources and summary candidates. The required minimal overlap, too, is a parameter that can be set by the user for creating a corpus; the lower the value, the more candidate summaries and source documents will be selected but the difficulty increases. In this paper, we use the number of shared bigrams to approximate the similarity. The quantity of overlap shows how much the summary and source texts contain similar concepts, but

[1] Wookieepedia, `https://starwars.fandom.com`

it can only be a first hint as to whether the information in the sources is sufficient to re-create the abstractive summary given a particular user-query. Therefore, in addition to the overlap, we create extractive summaries from the selected candidate sources based on the abstractive summary. An automatically calculated quality score for the extractive summary is used to select the set of summaries and source documents to form the final corpus.

In addition to the user-tuneable parameters of the fully-automatic process, users can also specify preferences as to which contents are particularly relevant. The Fandom wikis, for example, use a category system, as most wikis do. As a default, articles from all categories are extracted, but it is possible to restrict the categories, e.g., to discard all articles about non-fictional characters (i.e. actors, directors, film crew, ...) from a corpus about a movie franchise.

**Selecting Summaries for the Corpus:** The heuristics mentioned above help to identify possible candidates for triples consisting of a summary, a list of source documents, and a query—however, their quality can vary significantly as we show in our evaluation in Section 13.4. For some of them, the summary is indeed a high-quality summary of the extracted documents complying with the query, while for others it is hardly possible to find the information of the summary in the source documents. Hence, in a final step, we need to identify the usefulness of each triple and select only those which exceed a predefined quality threshold.

The selected summaries can optionally be split into training, validation, and test set. The split sizes—like all other parameters of the pipeline—can be adapted by the user of the framework according to their needs.

### 13.2.2. Extractive Summaries for Final Selection

Building an extractive summary involves choosing the best subset of sentences from the sources that form a summary of their content. In this paper, the extractive summarization procedure is modelled as an Integer Linear Program (ILP) based on the ideas of Boudin, Mougard, and Favre [BMF15], and Avinesh P. V. S. and Meyer [AM17]. The main intuition is that the ILP extracts the sentences with the most important concepts from the source documents to form a summary within a maximal length. To model the importance of sentences, we weight concepts according to their frequency in the human-written text (i.e., the selected candidate summary from the Fandom wiki). By doing so, we reward the system for a summary that contains many concepts of the abstractive reference summary. We use bigrams as concepts and ignore those consisting solely of stopwords.

To find good candidate triples, we use the objective score of the ILP for extractive approximation of the summary. This score is high if the extractive summary contains many concepts from the reference summary, hence resembling it well. For the final corpus, we only use summaries with a score higher than a certain threshold. In our evaluation in Section 13.4.2, we show how different values for this threshold impact the overall corpus quality.

In this paper, we use two different optimization objectives for the ILP. In both formulations, $c_i$ refers to the individual concepts and $L$ to the maximal summary length (which we set according to the selected range of the target length for the abstractive summaries). Moreover, sentences are referred to as $s_j$ with length $l_j$ and $Occ_{ij}$ meaning that concept $c_i$ occurs in that sentence. The first ILP formulation, as shown below, intends to maximize the overall sum of weights for distinct

covered concepts, while making sure that the total length of all selected sentences stays below a given threshold and the weight of a concept is only counted if it is part of a selected sentence.

$$\begin{aligned}
\max \sum_i w_i c_i \\
\forall j. \sum_j l_j s_j \leq L \\
\forall i, j. s_j Occ_{ij} \leq c_i \\
\forall i. \sum_j s_j Occ_{ij} \geq c_i \\
\forall i. c_i \in \{0, 1\} \\
\forall j. s_j \in \{0, 1\}
\end{aligned}$$

The second objective is simpler and tries to maximize the weight of the distinct selected sentences. Therefore, it is rewarded if an important concept appears in multiple sentences.

$$\begin{aligned}
\max \sum_j s_j \sum_i w_i Occ_{ij} \\
\forall j. \sum_j l_j s_j \leq L \\
\forall j. s_i \in \{0, 1\}
\end{aligned}$$

In our experiments, we evaluate both of these ILP formulations with regard to the final corpus quality. Both approaches use only syntactical features and no semantic ones (e.g., embeddings). They do not require time-intensive training and can be computed within a few seconds. Yet, it would be easy to exchange this component of the pipeline, if needed for a certain application.

## 13.3. Properties of Our Corpora

In the previous section, we have presented our new approach for automatically constructing summarization corpora. Using this approach, we have created three different sample corpora (one for Harry Potter, two for Star Wars) using the Fandom wikis as input. In this section, we will now discuss the unique properties of these corpora which differentiate them from other available corpora and, thus, make them a valuable contribution on their own. These sample corpora are all available for download with the sources of our construction pipeline.

First of all, our corpora do not feature news texts with their typical peculiarities (e.g., all important sentences at the beginning) but a mix of encyclopedic and narrative (story-telling) texts. In contrast to other sources, in Fandom wikis there are not a few dozens but thousands of articles about a certain topic. If the corpus is constructed from a single community, all articles are from the same domain (i.e., a closed world). However, it is also possible to utilize the common structure of the

| Corpus | Star Wars (en) | Star Wars (de) | Harry Potter | |
|---|---|---|---|---|
| Quality Threshold | 50 | 50 | 50 | 20 |
| # Articles | 148,348 | 39,356 | 15,993 | 15,993 |
| Candidates | 5,659 | 999 | 1,466 | 1,466 |
| Selected Summaries (train/valid/test) | 882 / 109 / 107 | 221 / 28 / 28 | 205 / 23 / 26 | 1,171 / 146 / 147 |
| Avg. Summary Length | 261.14 | 270.79 | 270.06 | 245.47 |
| Avg. # of Source Docs per Summary | 24.69 | 20.15 | 19.79 | 17.11 |
| Avg. Source Length per Doc | 1,143 | 855 | 3,087 | 3,400 |
| Avg. Overall Source Length per Summary | 28,236 | 17,241 | 61,111 | 58,188 |

Table 13.2.: Properties of the three sample corpora. For each, the amount of textual documents, the amount of candidates for a topic (target summary and matching source documents), and the amount of documents selected by the quality threshold (split into train, validation and test sets) are reported. For the Harry Potter corpus, the sizes of a second variant with a lower quality threshold are listed. For the selected summaries, the average length of the summary (in words) as well as the average size of the input documents (in words) and their average number per summary are reported.

different communities and build a corpus containing texts of different domains, e.g., to train more general summarizers.

Additionally, new corpora are fast and cheap to construct with just a minimum of manual work needed. There are many communities with lots of articles (e.g., Star Trek with 47,181 articles, Dr. Who with 71,425 articles) and the wikis are still growing. Moreover, communities are available in many different languages, hence this approach can be used to create corpora for various languages (e.g., one of our sample corpora is in German). The Creative Commons License of the texts allows us to offer the resulting corpora for download instead of only publishing tools for re-creating the corpora. This is in contrast to many existing news-based corpora such as Zopf [Zop18] which depend on crawling and thus the availability of external resources.

Last but not least, the abstractive texts in our corpora are of high quality since they are written by volunteers with intrinsic motivation and not by poorly paid crowd workers rushing through the task. A sample for such an abstractive text which shows the high-quality can be seen in Figure 13.4.

## 13.4. Analysis & Results

In our analysis, we show the validity of our pipeline and the usefulness of the generated data using three sample corpora created with our approach (two in English, one in German). We start by analyzing the properties of the automatically constructed corpora, then discuss the design decisions and validity of our pipeline steps, and finally run state-of-the-art summarizing systems on the data and evaluate their performance.

### 13.4.1. Statistics of Corpora

As a first analysis, we computed several statistics about the three sample corpora we constructed using our pipeline. The goal is to show whether, from a purely statistical perspective, the automatically constructed corpora are similar to manual (human-created) ones.

The results can be found in Table 13.2. The abstractive summaries have an average length of 260–270 words, the extractive summaries were created using a target length of 250 words, hence they have an average length little below that value. This length is similar to traditional multi-document summarization corpora like the DUC '06 and DUC '07 datasets[2]. It is a lot longer than the average length of 50 words of the live blog corpus [APM18] and drastically longer than the headline summaries of multiple news-based corpora such as Gigaword [NGV12].

The average number of source documents per summary lies between 19 and 25 documents. This as well is similar to the DUC '06 and DUC '07 datasets, higher than the ten documents considered in the DUC '04 and TAC '08[3] challenges, and about one half to one fourth of the amount of snippets per summary for the live blog corpora. The average length of the source documents is one to two magnitudes higher than for the live blog corpora, resulting in a higher overall source length to extract the important concepts from. Especially for Harry Potter, the overall length is two to three times higher than for the other two corpora, making this task especially hard.

The size of the final corpus varies depending on the size of the Fandom community and the quality threshold. For our sample corpora, it ranges from 250 topics, which is similar to the DUC '06 dataset used for traditional summarization approaches, to 1,300 topics, which is a size that can be used to train deep learning approaches. Additionally, it is possible to combine topics from multiple communities into a single training corpus.

This has an effect on the domain distribution and topic heterogeneity as well. A corpus constructed from a single community covers topics from only one domain, with the main difference between documents being whether they are about an event, a place, a being or a thing. Mixed corpora may contain texts from totally different domains (e.g., about a movie, a video game and baking recipes). The heterogeneity of writing styles, levels of detail, narrating styles and more, comes from the nature of the wiki itself and is inherently contained in all of the corpora.

In summary, it can be seen that, from a statistical perspective, it is possible to generate corpora with various properties matching typical needs of current (multi-)document summarization tasks.

### 13.4.2. Validation of the Pipeline

Our pipeline requires some parameters. Most of them are straightforward and can be adapted directly, according to the task at hand (e.g., the target length of the summaries) or have a direct impact on the difficulty of the dataset (e.g., the range of the amount of source documents or the length ratio between source and target). The most important parameter is the quality threshold (and connected to it the method to generate a score for the *extractability* of the summary from the sources). In this section, we evaluate how this parameter influences the overall corpus quality.

[2] `https://duc.nist.gov/`
[3] `https://tac.nist.gov/2008/`

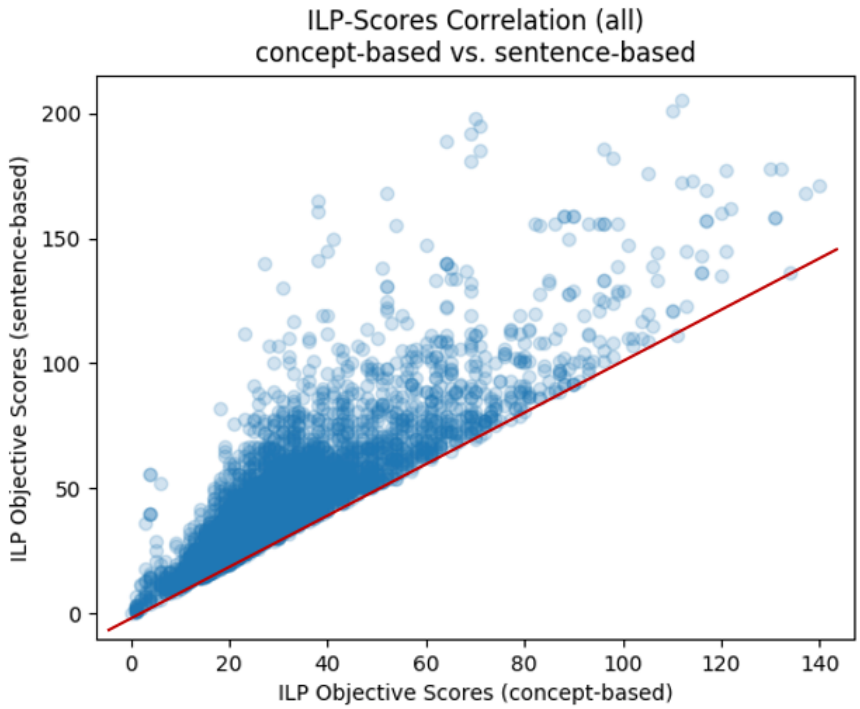



Figure 13.2.: Correlation between the objective scores of concept- and sentence-based extraction method on the same data.

First, we compare the two extraction modes (i.e., the two different ILPs described in Section 13.2). Figure 13.2 shows the correlation between the scores of both methods. It can be seen that the score of the sentence-based method is always equal or higher than the one of the concept-based method on the same data. The reason is that the sentence-based method will always get at least the same score for the summary selected by the concept-based method, but the score will be higher as soon as at least one duplicate concept is in it. However, normally both approaches produce similar but nonetheless different summaries, and the quality of a summary produced by the concept-based method might be better than a sentence-based one with a higher score, because for the first one duplicate concepts were not rewarded.

The correlation justifies using either of the two methods as a quality indicator. However, the question how the quality of the summary really correlates to the score of the extraction remains. To assess this, we asked human annotators to evaluate the quality of 39 equally distributed summaries. We asked them to decide for each sentence in the human abstract if it is covered by the extractive summary (0) not at all, (1) partially, (2) mostly, or (3) fully. The human decision is averaged for the full summary and correlated to the score of the extraction. The results can be found in Figure 13.3. It can be seen that a higher ILP score does indeed correlate with a better human evaluation. Based on this we have chosen the ILP-thresholds for the selection of the summaries. Our experiments suggest a value of about one fifth of the target length ($250/5 = 50$) for the sentence-based method, while for the concept-based method the corresponding threshold would be slightly lower since duplicates are not counted.

In addition, we show a sample for a human abstract and the corresponding extractive summary in Figure 13.4, to intuitively demonstrate that extractive summaries are a good stub to judge

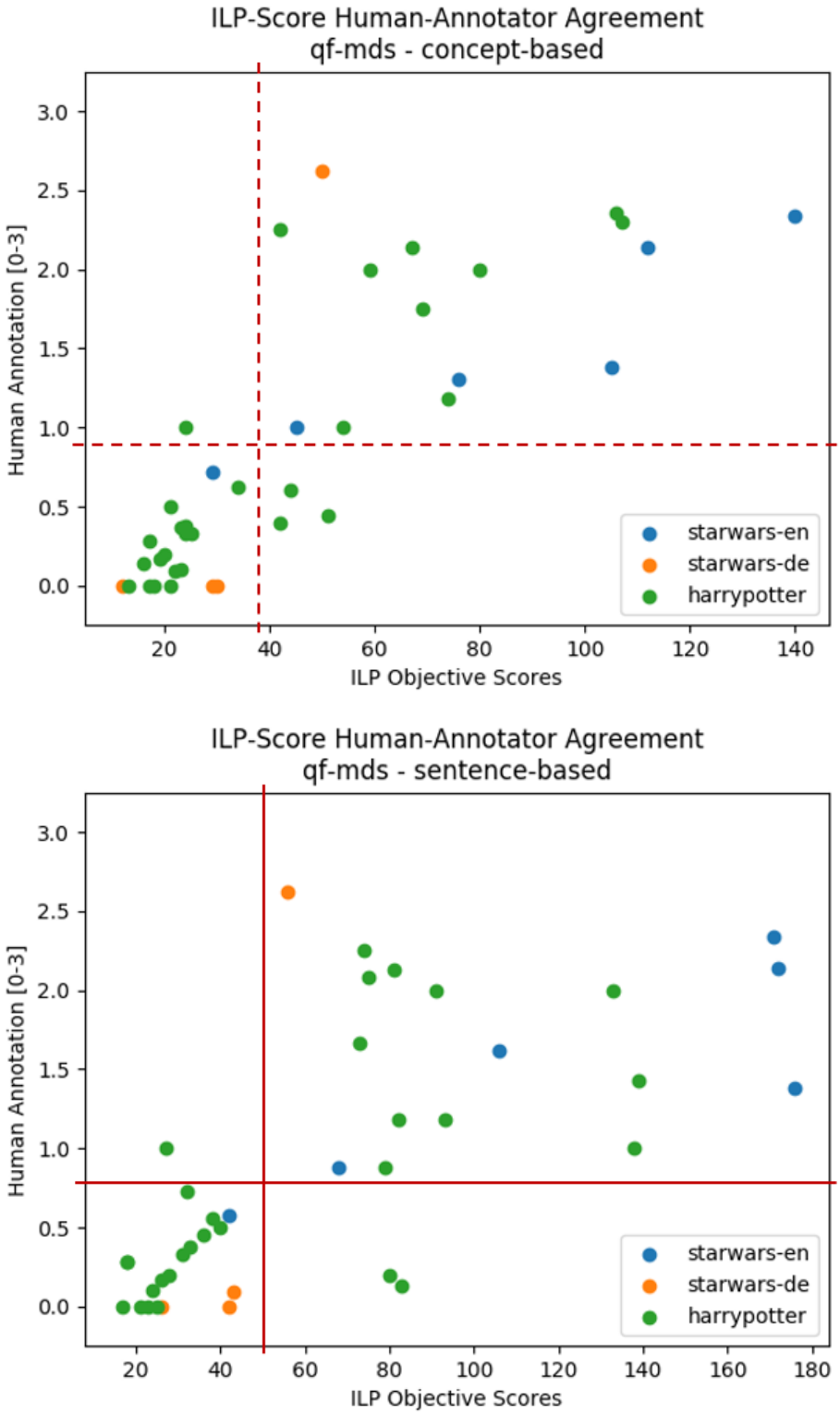



Figure 13.3.: Human agreement with automatically generated extractive summaries for concept-based and sentence-based creation method. Average values for the sentences of 39 annotated documents, possible values between 0 and 3 (best).

**Human-written abstract**

At Hogwarts School of Witchcraft and Wizardry, sixth years are typically 16 to 17 years of age, although some may be older, if they have had to repeat a year like Marcus Flint. The sixth year is the year in which students advance to N.E.W.T.-level classes. [...] Neither the core classes nor the elective courses are available to any student who does not meet said requirements. While students do have the opportunity to choose whether they wish to continue in particular subjects, those who begin studying N.E.W.T.-level subjects in their sixth year are expected to carry on with the subject into the seventh year, and sit the N.E.W.T. exam in that subject. [...] Students in the sixth year may also elect to take part in Apparition lessons for a fee of twelve Galleons. [...]

**Extractive summary**

The fifth year is also the year in which students receive career counselling from their Heads of House. Sixth years are typically sixteen to seventeen years of age, although some may be older, if they have had to repeat a year like Marcus Flint did. Sixth years may also elect to take part in Apparition lessons for a fee of twelve Galleons. [...] Depending on the minimum requirements of the professor teaching the subject at that time, students are allowed to sit any number of classes as long as they meet said requirements. While students do have the opportunity to choose whether they wish to continue in particular subjects, those who begin studying N.E.W.T.-level subjects in their sixth year are expected to carry on with the subject into the seventh year and sit the N.E.W.T. exam in that subject. [...]

Figure 13.4.: Sample human abstract and the corresponding auto-generated extractive summary (concept-based) for topic "Sixth year: During the sixth year" (00943) from the Harry Potter corpus. Due to spacing reasons, only excerpts are shown. From 10 sentences in the human abstract, 7 are fully covered by the extractive summary, one sentence mostly and two sentences not at all, leading to a human evaluation score of 2.3 with an optimization score of 107.6 for the extraction ILP.

the quality of source documents for abstractive summarization. While we can see that a good quality of the extractive summary implies that the source documents are useful for abstractive summarization of the given documents, there is still room for improvement: First, some sentences might be missing in the extractive summary simply because the length of the extractive summary is typically lower than the abstractive one (since not the full length could be exploited). A second problem can be found in the Harry Potter wiki, but it is likely that it will frequently occur in other domains as well: in many cases all names of people or places in a summary are linked to articles about them, adding these articles to the source documents. Yet, without co-reference resolution and explicit query handling, the system is prone to selecting sentences about the wrong entity as input. More generally speaking, there is a lack of real understanding of the extracted contents in our construction pipeline. The approach works solely on a syntactic level and does not use any semantic features such as synonyms at the moment. We want to address this as an extension of our pipeline in future work.

### 13.4.3. Corpora Quality

In these experiments, we ran multiple well-known techniques that were successfully used for single and multi-document summarization. The goal is to show that the quality of the automatically created corpora is high enough that state-of-the art summarizers can perform reasonably well on those corpora. Our implementations are based upon the implementation by Avinesh P. V. S., Peyrard, and Meyer [APM18]. For the assessment of summary quality based upon a reference summary, we compute and report the ROUGE metrics. Owczarzak et al. [Owc+12] show that

| Systems | Harry Potter | | | Star Wars (en) | | | Star Wars (de) | | |
|---|---|---|---|---|---|---|---|---|---|
| | R1 | R2 | SU4 | R1 | R2 | SU4 | R1 | R2 | SU4 |
| **Luhn** | 0.1669 | 0.0308 | 0.1366 | 0.2440 | 0.0523 | 0.2045 | 0.1725 | 0.0357 | 0.1378 |
| **LexRank** | **0.3702** | **0.0729** | **0.2850** | **0.3845** | **0.1049** | **0.3103** | **0.3579** | **0.0784** | **0.2711** |
| **LSA** | 0.3113 | 0.0421 | 0.2454 | 0.3135 | 0.0533 | 0.2550 | 0.3081 | 0.0512 | 0.2350 |
| **KL** | 0.2407 | 0.0528 | 0.1897 | 0.3087 | 0.0808 | 0.2546 | 0.2213 | 0.0524 | 0.1742 |
| **ICSI** | 0.2224 | 0.0360 | 0.2041 | 0.3041 | 0.0423 | 0.2507 | 0.2199 | 0.0353 | 0.1984 |
| **UB1** | **0.5585** | 0.1744 | 0.3802 | **0.5793** | 0.2341 | 0.4210 | **0.6095** | 0.3354 | 0.4859 |
| **UB2** | 0.5465 | **0.2609** | **0.4137** | 0.5700 | **0.3050** | **0.4491** | 0.6089 | **0.3847** | **0.5111** |

Table 13.3.: Average scores (ROUGE-1, ROUGE-2, ROUGE-SU4) for different baseline systems on all candidates of all three sample corpora. Values between 0 and 1, higher is better.

| Systems | Harry Potter | | | Star Wars (en) | | | Star Wars (de) | | |
|---|---|---|---|---|---|---|---|---|---|
| | R1 | R2 | SU4 | R1 | R2 | SU4 | R1 | R2 | SU4 |
| **Luhn** | 0.1791 | 0.0365 | 0.1475 | 0.2605 | 0.0560 | 0.2195 | 0.1830 | 0.0412 | 0.1491 |
| **LexRank** | **0.3855** | **0.0881** | **0.3053** | **0.3929** | **0.1083** | **0.3227** | **0.3662** | **0.0849** | **0.2849** |
| **LSA** | 0.3267 | 0.0545 | 0.2635 | 0.3293 | 0.0584 | 0.2722 | 0.3226 | 0.0624 | 0.2541 |
| **KL** | 0.2753 | 0.0655 | 0.2176 | 0.3116 | 0.0780 | 0.2609 | 0.2321 | 0.0617 | 0.1902 |
| **ICSI** | 0.2440 | 0.0419 | 0.2245 | 0.3223 | 0.0496 | 0.2683 | 0.2350 | 0.0412 | 0.2125 |
| **UB1** | 0.6261 | 0.2885 | 0.4742 | 0.6830 | 0.4115 | 0.5656 | 0.7513 | 0.5811 | 0.6815 |
| **UB2** | **0.6265** | **0.3746** | **0.5122** | **0.6835** | **0.4726** | **0.5939** | **0.7569** | **0.6164** | **0.7027** |

Table 13.4.: Average scores (ROUGE-1, ROUGE-2, ROUGE-SU4) for different baseline systems on selected summaries (score of sentence-based extraction ILP greater or equal to the quality threshold of $50$) of all three sample corpora. Values between 0 and 1, higher is better.

these metrics strongly correlate with human evaluations of this similarity. We report the ROUGE-1 (R1) and ROUGE-2 (R2) metrics (without stemming or stopword removal) as well as ROUGE-SU4 (SU4) as the best skip-gram matching metric.

**Baseline Summarizers:** As state-of-the-art summarizers, we use the following systems:

*TF*IDF* [Luh58]: The sentences are scored with the term frequency times the inverse document frequency for all their terms, ranked by this score and greedily extracted.

*LexRank* [ER04]: This well-known graph-based approach constructs a similarity graph $G(V, E)$ for all sentences $V$ with an edge between them if their cosine-similarity is above a certain threshold. The summary is built by applying the PageRank algorithm on this graph and, again, extracting greedily.

*LSA* [SJ04]: This approach uses singular value decomposition to reduce the dimensions of the term-document matrix to extract the sentences containing the most important latent topics.

*KL-Greedy* [HV09]: This approach tries to minimize the Kullback-Leibler (KL) divergence between the word distributions of the summary and the source documents.

*ICSI* [GF09]: This approach is based on global linear optimization. It extracts a summary by solving a maximum coverage problem that considers the most frequent bigrams in the source documents. Hong et al. [Hon+14] found this to be among the state-of-the-art systems for multi-document summary.

We applied all of these approaches to all topics of our corpora. Due to large input sizes, LexRank, LSA, KL-Greedy and ICSI did not terminate in a reasonable time on some topics. The affected topics varied for each approach.

In addition, to judge the quality of the baselines, we also computed the upper bound that an extractive summarizer could achieve in the best case. An extractive summarization system normally cannot re-create the human-written abstractive text exactly, since the abstractive sentences differ from the sentences of the source texts that can be extracted. Hence, the best overlap between an abstractive and the best extractive text is usually below 100%. To take this into consideration, we compute and report those upper bounds for extractive systems as suggested by Peyrard and Eckle-Kohler [PE16]. This is done using the first ILP from Section 13.2.2 with slightly adapted concepts and weights: we compute one upper bound based on unigrams (UB1) and one upper bound based on bigrams (UB2). For both of them, the concepts are not weighted but the maximum coverage of distinct n-grams is counted. As for the baselines, the ROUGE scores for the created extractive summary compared to the abstractive text are computed.

**Neural Summarizers:** In addition to the baseline systems mentioned above, we also evaluate the data using learned models. To do so, we use the best scoring model combination for extractive summarization by Kedzie, McKeown, and Daumé III [KMD18], a combination of a Seq2Seq model as extractor and an Averaging Encoder. Yet, our datasets use a compatible data format, hence all other models evaluated in that paper can be used on our data as well[4]. For training, the extractive summary provides a binary decision for every input sentence, i.e. whether it should be part of the

[4]Kedzie, McKeown, and Daumé III [KMD18] provide reference implementations with an unified interface for all evaluated models at
`https://github.com/kedz/nnsum/`

| Corpus | Extraction Mode | Quality Threshold | Validation | Test: Human Abstracts | | | Test: Auto-Extractive | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | R2 | R1 | R2 | SU4 | R1 | R2 | SU4 |
| Star Wars (en) | concept | 50 | 0.0876 | 0.4461 | 0.1501 | 0.4148 | 0.4729 | 0.1807 | 0.3369 |
| | sentence | 50 | 0.0864 | 0.4375 | 0.1476 | 0.4042 | 0.4721 | 0.2096 | 0.3310 |
| Harry Potter | concept | 50 | 0.0655 | 0.3557 | 0.0714 | 0.3321 | 0.3754 | 0.0876 | 0.2528 |
| | sentence | 50 | 0.0692 | 0.3528 | 0.0699 | 0.3290 | 0.3657 | 0.0802 | 0.2473 |
| | sentence | 20 | 0.0460 | 0.3673 | 0.0690 | 0.3384 | 0.3775 | 0.0888 | 0.2562 |
| Star Wars (de) | concept | 50 | 0.1127 | 0.4197 | 0.1700 | 0.3984 | 0.4189 | 0.1606 | 0.3066 |
| | sentence | 50 | 0.1294 | 0.4365 | 0.1852 | 0.4146 | 0.4456 | 0.2086 | 0.2957 |
| Combined (en) | concept | 50 | 0.0749 | 0.4136 | 0.1151 | 0.3825 | 0.4500 | 0.1501 | 0.3126 |
| | sentence | 50 | 0.0753 | 0.3927 | 0.0947 | 0.3629 | 0.4019 | 0.1079 | 0.2729 |

Table 13.5.: Average ROUGE values for Seq2Seq models (neural baseline) trained on the different training sets and tested on the original human abstracts and the auto-generated extractive abstracts of the respective test sets. Rouge values between 0 and 1, higher is better.

summary or not. For generation, a probability is inferred for every sentence and then used to rank them and extract greedily.

We benchmark all three corpora with both extraction methods and a quality threshold of 50. Additionally, we run the benchmark on the Harry Potter corpus with sentence-based extraction and a threshold of 20, and on a combined dataset (Star Wars, Harry Potter and Star Trek[5], all English). All experiments use 200-dimensional GloVe vectors to represent words.

**Analysis of the Summarization Quality:** Table 13.4 shows the benchmark results of the selected summaries for the three sample corpora. We report the ROUGE-1, ROUGE-2 and ROUGE-SU4 scores for the different baseline systems. All experiments use a target length of 250 words, if not stated otherwise. This corresponds to the length of the commonly used DUC '06 and DUC '07 datasets. When compared to the benchmark runs on all candidates of the corpora (see Table 13.3), one can see that the average scores for all systems are higher on the selected summaries, proving that these are, on average, better pairs of summary and source documents. However, in relation to the upper bounds (UB1 and UB2), even the best performing baseline (LexRank) can only reach one third to one fifth of the upper bound on ROUGE-2 (for ROUGE-1 and SU4 it is at least half or better). This reflects our findings from other papers, e.g., Avinesh P. V. S. and Meyer [AM17], and thus we believe that the quality of our automatically constructed corpora is on par with the manually created ones used in previous evaluations. Moreover, the fact that state-of-the-art summarizers can only reach one third to one fifth of the upper bound on ROUGE-2 also emphasizes that multi-document summary is still a challenging task in general and needs further research which we hope to stimulate with this paper.

This is also stressed by the following findings: Table 13.5 shows the results of training multiple sequence-to-sequence models with the training data from the corpora. We tested them both on the original human abstracts and the extractive summaries that were automatically created based on them to see the effect of the abstraction. The scores on the extractive test set are higher for all models, as expected. The test on the human abstracts can be compared to the results of the

[5] `https://memory-alpha.fandom.com/`

non-neural baselines from Table 13.4. We can see that the neural approach outperforms the other baselines on the Star Wars corpora. Especially for the German variant the result is surprisingly good even though we are not using German embeddings but rather standard GloVe vectors. However, for the Harry Potter corpus, the neural baseline cannot even outperform the LexRank baseline. We find three reasons for that: first, the total length of the source documents (which is two to three times higher than for the other two corpora), second the linking style of the wiki (see Section 13.4.2) and third the comparatively low amount of training data. It can be seen that scores for the model trained on the variant with lower quality threshold (leading to a five times higher corpus size) are similar or even higher. Getting similar results from training data of a worse quality supports the assumption that the amount of training data is a problem here.

For those cases, we test a combined corpus, where texts from multiple domains are combined. We can see that this can be used to handle the lack of training data, but that a specialized model will outperform this more general model when there is enough training data available.

## 13.5. Future Work

With this paper, we present ready-to-use data and an approach to generate more. Of course, there is still room for improvement and extensions of the pipeline:

One important point is the generation of extractive summaries. As discussed in Section 13.4.2, our pipeline does not exploit semantic features yet. The use of semantic word representations, word sense disambiguation, co-reference resolution, or entity linking could create better extractive summaries and serve as a better basis for quality threshold decisions.

A second important point is the length of the source documents to be summarized. Since wiki authors are encouraged to add a lot of hyperlinks between the texts, the list of source documents might contain articles not entirely relevant for the topic, making it very hard to solve the summarization task. Future work should focus on developing methods to choose more relevant subsets of the source texts. Semantic features could play an important role here as well.

Finally, we think that our approach can also work as a basis to generate data for other tasks. One example is hierarchical summarization [Chr+14]: Fandom communities about television series often contain articles about every single episode, about each season and articles about certain aspects of the full series. These articles all have different levels of detail and form a hierarchy that can be extracted using some simple manual rules.

## 13.6. Conclusion

In this paper, we presented a novel automatic corpus construction approach and three open-licensed corpora for multi-document summarization based on this approach. All corpora are available online to be used directly by other researchers, together with a ready-to-use framework to create custom corpora with desired parameters like the length of the target summary and the amount of source documents to create the summary from. We verified the pipeline and showed the usefulness of the corpora, including the fact that state-of-the-art summarizers cannot yet solve all challenges posed by our new corpora. Our data could contribute to the further development of systems for

(semi-)automatic multi-document summarization, especially those exploiting the query or relying on user feedback. The framework can be used to generate further training and test data for these systems or serve as basis to generate data for other tasks, such as hierarchical summarization.

# 14. It's AI Match: A Two-Step Approach for Schema Matching Using Embeddings (AIDB'20)


## Abstract

Since data is often stored in different sources, it needs to be integrated to gather a global view that is required in order to create value and derive knowledge from it. A critical step in data integration is schema matching which aims to find semantic correspondences between elements of two schemata. In order to reduce the manual effort involved in schema matching, many solutions for the automatic determination of schema correspondences have already been developed.

In this paper, we propose a novel end-to-end approach for schema matching based on neural embeddings. The main idea is to use a two-step approach consisting of a table matching step followed by an attribute matching step. In both steps we use embeddings on different levels either representing the whole table or single attributes. Our results show that our approach is able to determine correspondences in a robust and reliable way and compared to traditional schema matching approaches can find non-trivial correspondences.


## Bibliographical Information


The content of this chapter was previously published in the peer-reviewed work "Benjamin Hättasch, Michael Truong-Ngoc, Andreas Schmidt, and Carsten Binnig. 'It's AI Match: A Two-Step Approach for Schema Matching Using Embeddings'. In: *2nd International Workshop on Applied AI for Database Systems and Applications (AIDB20). In conjunction with the 46th International Conference on Very Large Data Bases, Virtual, August 31 - September 4, 2020*. Virtual, 2020. DOI: 10.48550/arXiv.2203.04366". The contributions of the author of this dissertation are summarized in Section 5.1.

## 14.1. Introduction

**Motivation:** The quality of decisions is directly influenced by the available data and the ease of access to it. Important decisions should thus be made based on a holistic view using all available data. However, data is often scattered into different heterogeneous sources. This is not only the case for many data science projects that need to integrate data from different independent sources but is also relevant within companies, where data typically resides in different systems. Data Integration can help to mitigate these issues since it allows to create a global view over independent data sources.

A critical step in data integration is schema matching which aims to find semantic correspondences between elements of two schemata. Traditionally, this was done by experts with a good understanding of the semantics of the data [Hul97]. However, modern schemata are becoming larger and more complex, making manual schema matching both more time-consuming and more error-prone [RB01]. In order to reduce the manual effort involved in schema matching, many solutions for the automatic determination of schema correspondences have already been developed [Che+12; DR02; Geo05; LN07; MIA17; NHN19; PKT09; Pin+15; RB01; SMJ19].

**Contribution:** A major problem of many automatic schema matching approaches is that they fail if the semantic similarity is hard to detect. For example, instance-based column matchers typically fail to match columns that contain disjoint but semantically similar values such two tables with different street names or even worse the same content in different languages (e.g., French and English). Another example are name-based matchers that rely on sources such as *WordNet* to identify column matches: while these approaches can detect hard-to-match cases (such as columns that use synonyms as names), they fail if this knowledge is not encoded in the resource.

In this paper, we thus propose a novel end-to-end approach for schema matching based on neural embeddings to mitigate these issues. The main idea is to use a two-step approach consisting of a table matching step followed by an attribute matching step. In both steps we use embeddings on different levels (i.e., representing the whole table but also only single attributes). This allows our approach to find non-trivial correspondences such as those discussed before.

Summarized the main contributions of this paper are:
(1) First, we provide an analysis of existing approaches to automatically support schema matching. (2) We then present our end-to-end approach for schema matching using neural embeddings. (3) We propose and analyze different matchers on multiple levels (i.e., tables and columns) to identify a set of possible table and attribute correspondences. (4) We evaluate our approach on several benchmarks and real-world data sets. Our results show that our approach is able to determine correspondences in a robust and reliable way and compared to traditional schema matching approaches can find non-trivial correspondences.

**Outline:** This paper is structured as follows: In Section 14.2 we give an overview about existing approaches for schema matching. Afterwards, in Section 14.3 we show how neural embeddings can be used for schema matching in general and which challenges and problems can arise. In Section 14.4 we give an overview of our two-step approach, which will be further elaborated and evaluated in Sections 14.5 and 14.6. Finally, in Section 14.7, we summarize our results and give an outlook on possible further work.

## 14.2. Previous Approaches

Conventional supporting approaches for schema matching use syntactical features like *Levenshtein distance* or *n-grams* to compute the similarity between different schema information. More recent approaches also try to consider semantic aspects such as synonyms and hypernyms or rely on machine learning (ML). In the following, we give an overview of the existing matching approaches and then discuss their shortcomings compared to our approach.

There already exist multiple matching frameworks that integrate many of the before mentioned matchers or even combine them in so-called hybrid matchers. One prominent example is the COMA/COMA++ framework by Do and Rahm that provides reference implementations for various approaches [DR02]. Due to space constraints, however, we are not discussing the details of these matching frameworks in this paper.

### 14.2.1. Schema-Based Approaches

Schema-based approaches, as the name already implies, try to derive similarities from the available schema information such as tables and attribute identifiers, available comments and data types as well as constraints.

Islam and Inkpen [II08] present a method which derives the similarity of two texts by a combination of semantic and syntactic information. Their approach initially determines syntactic similarity using a modified version of the Longest Common Subsequence (LCS) metric, which provides an alternative method for the edit distance. Subsequently, the semantic similarity is determined corpus-based. They also propose an optional word order similarity calculation, which classifies two texts as similar if common words have the same order in their respective text. The final result for the similarity of two texts is then derived by normalizing and combining the three similarity metrics. This method has the disadvantage that a significantly large corpus of schema similarities must be available in order to learn enough semantic knowledge.

The approach of Lee et al. [Lee+09] compares the semantics of attributes to overcome semantic heterogeneity in the healthcare domain. It consists of two components: word similarity and word affinity. The similarity of two words is determined using domain-specific knowledge and *WordNet* (synsets). The affinity of two words is deduced from overlapping characters, hyponymy and hyperonymy relationships.

Chen et al. [Che+12] present a hybrid algorithm that determines correspondences using concepts. To extract the concepts, again *WordNet*-synsets as well as the tree structure of the hyperonymy-hyponymy relationship is used. Afterwards, concepts are compared both syntactically (name) and semantically (WordNet node) and conceptual relationships at node level are transformed into conjunctive normal forms to confirm semantic relationships and derive similarity values.

### 14.2.2. Instance-Based Approaches

In contrast to schema-based approaches, instance-based approaches attempt to derive similarities between attributes from the values belonging to them.

Partyka et al. [PKT09] examine the values of the attributes to be compared and try to derive their similarity to the *entropy-based distribution* (EBD).

While earlier approaches determined this distribution by means of n-grams, which are, however, highly dependent on overlapping instances, Partyka et al. take a different approach to determine the EBD. Their *TSim* algorithm uses the *normalized Google Distance* (NGD), together with the cluster method *K-Medoid*: Individual keywords are extracted from the instances and afterwards grouped by K-Medoid. EBD values for these clusters are then calculated by semantic comparisons of the instances with the NGD to derive attribute correspondences.

Mahdi and Tiun [MT14] combine strengths of regular expressions for numerical instances and semantic relations from *WordNet* for text-based instances to determine correspondences instance-based. Similarly, Mehdi et al. [MIA17] propose a combination of regular expressions and an external auxiliary source. Attributes are first classified as numeric, text-based, or mixed based on their instances. Samples of instances from each class are then extracted and compared. Regular expressions are applied to numeric and mixed instances to examine instances for syntactic similarities based on patterns. To measure the semantic similarity of text-based instances, the Google Similarity Distance (*Google Similarity Distance*) [CV07] is used.

### 14.2.3. ML-Based Approaches

*WordNet* and other lexical dictionaries are limited in their scaling and do not cover all terms. For this reason, Zhang et al. [Zha+14] present a name-based ontology matching approach based on word embeddings (vector representations of words, see next section for more details) of the *Word2Vec* model. A hybrid approach combining word embeddings and editing distances achieved the best results in their experiments.

The combined approach of Fernandez et al. [Fer+18] also relies on syntactic comparison methods and *Word2Vec*. They additionally present the concept of *Coherent Groups* for combining word embeddings for compound words. Their approach first determines correspondences using syntactic comparison methods such as the edit distance for attribute names and the Jaccard similarity for instances. Word embeddings for names are then used to both remove incorrect correspondences and add missing ones. In a comparison between the pre-trained *Word2Vec* and a domain-specific trained model, the former achieved better results.

The approach of Nozaki et al. [NHN19] is based purely on the word embeddings of *Word2Vec*, which are used to semantically compare text-based columns and to determine correspondences with sufficiently high semantic similarity. For this purpose, word vectors are generated for all instances of a column; instances consisting of compound words by the sum of the word vectors of their subwords, with subwords being ignored if they do not appear in the dictionary. The sum of all instance vectors finally forms the attribute vector.

Gentile et al. [Gen+17] present a method to reduce the search space for entity matching. The approach attempts to group similar tables by representing tables with *Word2Vec* vectors. They test three different approaches to generate a vector representation (table embedding) for a whole table: attribute model, where the embedding is generated using the table attributes, entity model, which is based on the so-called ”subject column” (a column of type string with the highest number of different values), and a combination of both. Their evaluation shows that table embeddings are a promising method for reducing the search space without explicit expert knowledge.

Chen et al. [Che+19] designed a hybrid neural network that is trained to predict the semantics of table columns without metadata and only with instances. Table contents are vectorized using word embeddings and fed into the neural network as input for training and prediction. In comparison to *Word2Vec*, their RNN-based embeddings could provide better results.

Sahay et al. [SMJ19] present a hybrid method that combines schema information and instance information to find similar attributes. A vector representation is generated using schema and instance features of a column. These vectors are then grouped using *k-means Clustering* and only columns within clusters are compared using syntactic comparison methods such as edit distance.

### 14.2.4. Discussion

This short summarization shows that research in the field of automatic schema matching already produced a lot of approaches. Hence, as mentioned before, simple schema-based or instance-based matchers have problems to discover semantic correspondences. More recent approaches based on machine-learning aim to address these shortcomings similar to our approach; some of those approaches already propose to use neural embeddings. Different from those approaches, however, a major novelty of our approach is that we take a two-step approach that uses neural embeddings at different levels (i.e., tables and columns) and also to combine different aspects (i.e., schema information and instances).

## 14.3. Using embeddings for Matching

In this section, we first will provide the necessary background to explain why neural embeddings (in particular word embeddings) provide a good starting point for schema matching.

### 14.3.1. Similarity based on Embeddings

In a nutshell, word embeddings represent words by vectors and map the—in many languages—unlimited number of words to a space with a fixed number of dimensions. One usually speaks of embeddings if the representation, in contrast to, for example, one-hot-encoding, contains information that goes beyond syntactic features, i.e. the words are not arranged arbitrarily in vector space, but rather from a semantic point of view. A typical possibility is to represent words by means of other words in whose context they often occur.

What makes word embeddings interesting for schema matching is that they can be used to compare semantic similarity between strings, hence improve over simple syntactic approaches like string-edit distances. In the case of schema matching, for example, attribute names can be represented by word embeddings. The similarity of two embeddings is computed by applying vector distance metrics, especially cosine similarity [Mik+13] which can be computed with low computing capacities using simple operations. Although vector components of word vectors of different word embeddings models can be negative, word vectors are often trained in such a way that the cosine similarity is between 0 and 1 in almost all cases and negative only in very rare cases [BP19]. Values near 0 indicate low semantic similarity, while values near 1 indicate high semantic similarity.

In contrast to simple string comparison, word embeddings can cover different syntactical variations for the same or similar concept (e.g., synonyms or abbreviations) and for contextualized word embeddings also different semantic meanings with the same syntax (i.e. ambiguities). However, there are also additional challenges: one difficulty is the merging of semantics and relationship of words [HRK15]. Words like *coffee* and *cup* are semantically not identical, but are often mentioned in the same context. This relationship is expressed in similar representations for many kinds of embeddings [Far+16].

### 14.3.2. Different Embedding Approaches

Word embeddings are usually trained on large corpora of text and therefore cover the relations between common words. The first generation of word embeddings like *Word2Vec* [Mik+13] or *GloVe* [PSM14] were explicitly created for comparing single words. There exist simply to obtain pre-trained models that can be used for this task without additional knowledge about there internal function or need for training [KR15]. Although these models were trained on large corpora (e.g., the English Wikipedia or Google News), their vocabulary is still limited: As a result, they do not support most of the multi-word expressions directly, and it is not possible to generate word representations especially for domain specific terms.

There are several approaches to generate embeddings for expressions consisting of multiple words like sentences or paragraphs [LM14; Soc+12]. However, such models do not work very well for combining attribute names or instances because the combination of table values does not form grammatically correct sentences [Fer+18].

An alternative to off-the-shelf sentence embedding models is to use the average vector of all individual words with pre-trained embeddings for single words. This method was developed by Castro Fernandez et al. [Fer+18] using *Word2Vec* for matching schema attribute names. They concluded that comparing the average vectors of multi-words is not an optimal solution for attribute matching and instead proposed *Coherent Groups* as an alternative method for comparing multi-words with *Word2Vec*. Coherent Groups consider multi-word expressions as groups of individual words. The method calculates the similarity of two groups of words, such as two attributes, by comparing the individual words in pairs. The average of all comparisons then determines the similarity between the two attributes.

Another problem are out-of-vocabulary (OOV) errors: traditional word embedding models can only represent words they have already seen during training, which works well for texts about common topics but may fail for databases with domain-specific element names and content. The amount of out-of-vocabulary words can be reduced by training or fine-tuning on domain specific texts. Yet this requires computing resources and suitable training corpora. The usage of embeddings only trained using the *Distributional Hypothesis* [Har54] (i.e. words are represented by the context they usually appear in) is therefore difficult for schema matching.

Contextualized word embeddings models like *ELMo* [Pet+18], *BERT* [Dev+18] or Google Universal Sentence Encoder (*USE*) [Cer+18] are trained for additional tasks like machine translation (using both suitable data and an adequate internal model architecture), so that the resulting word vectors are semantically more meaningful. Since they can also generate word vectors dynamically (i.e. find a representation for unseen words), no methods for the aggregation of multiple words and handling of OOV are required. This is done using WordPiece tokenization and breaking down unseen or rare

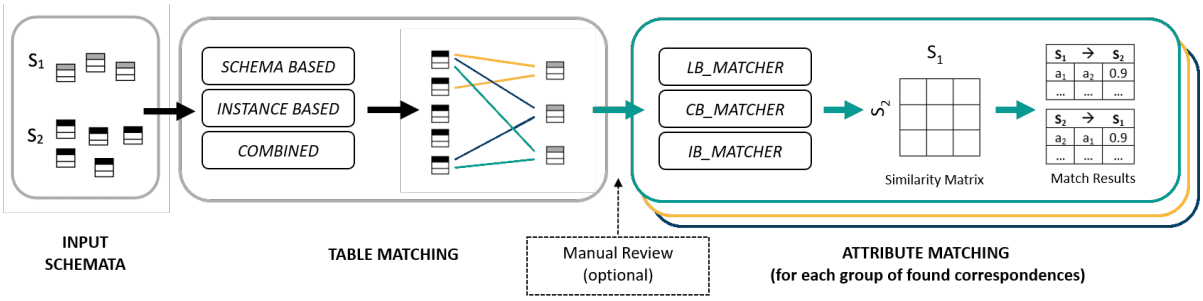



Figure 14.1.: End-to-end process for schema matching, consisting of two steps (table matching and attribute matching). In the first step, possible matches between the tables are produced either on the basis of the available schema information (`SCHEMA BASED`), or on the basis of the instances (`INSTANCE BASED`) of tables or using both together (`COMBINED`). Optionally, after the first step, a user can confirm or reject possible table matches. For the next step, matches between the individual attributes of the remaining table pairs are determined. This can be done using the table and attribute names (`NB_MATCHER`), natural language comments in the database schema (`CB_MATCHER`), or instances of a table (`IB_MATCHER`). Either thresholding or ranking can be applied to the resulting similarity values to determine the final matches.

words into subwords or characters. Therefore, partial semantic meanings (for segmentation into subwords) or at least syntactic information (for character level encoding) are preserved even for complex unknown terms and expressions and the downsides of the above-mentioned approaches can be avoided.

During the pre-training of these models, attention was paid to covering as much semantic variability and many domains as possible. Therefore, the pre-trained models should be sufficient for many application purposes. If, however, many domain-specific acronyms are used in a database (especially in the schema information), a preprocessing step with a domain-specific dictionary can be helpful. For example, it can be assumed that the generated word vector for *purchase order* is semantically more meaningful than that for *PO*.

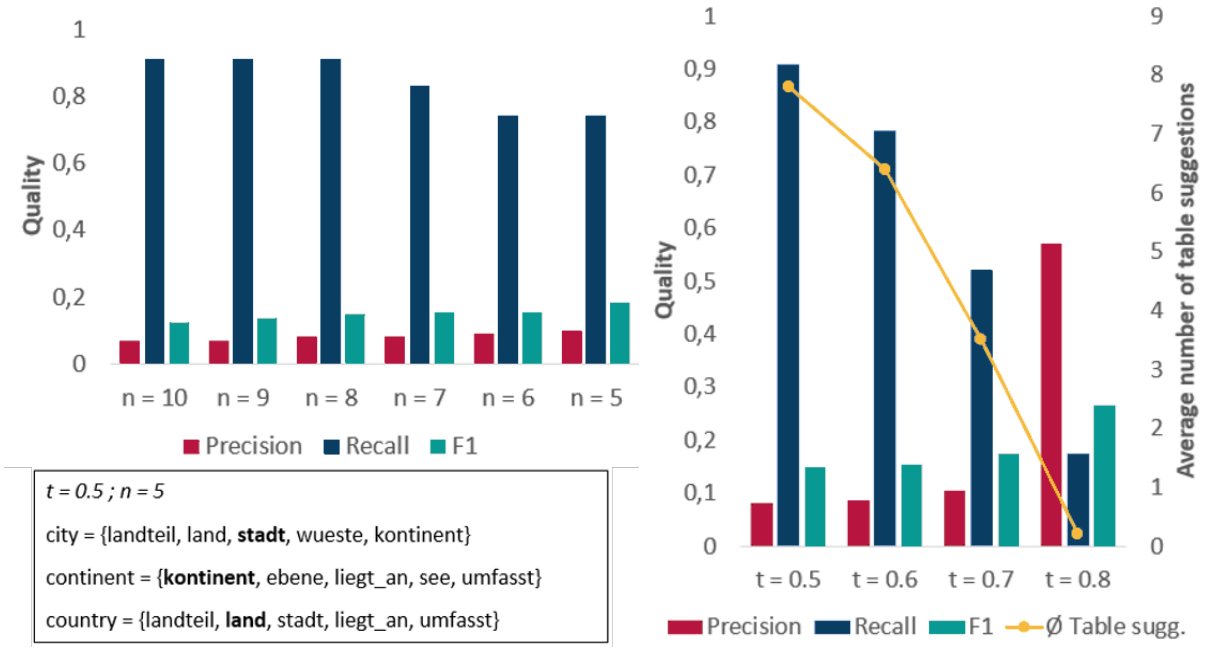
For the evaluation of our approach in the next sections, we will mainly use a multilingual *Google USE* model that was trained on 16 different languages.[1] Yet, the described approach works with any contextualized word embedding.

## 14.4. Overview of Our Approach

In the following, we present our novel end-to-end process for schema matching. As mentioned before, our approach consists of two steps that build on each other (table matching and attribute matching). For each step, we will propose and test different matchers based on neural embeddings. An overview of our approach can be seen in Figure 14.1.

While neural embeddings have been used already for individual tasks of schema matching (i.e., table or attribute matching), we suggest a new holistic approach that uses neural embeddings on

[1] `https://tfhub.dev/google/universal-sentence-encoder-multilingual-large/3`




(a) fixed $t = 0.5$ and varying $n$ values

(b) fixed $n = 8$ and varying $t$ values

Figure 14.2.: Experiment 1 – Cross-lingual schema-based table matching on geographic databases with different parameter settings. Precision, recall and F1 metric are reported. For a) a qualitative analysis of one setting is shown, for b) the average number of found possible correspondences for different similarity thresholds $t$ can be found.

different levels to combine table and attribute matching. To be more precise, in a first step, we use neural embeddings to match the elements of a schema on the table level. For each table in the target schema, we propose either all tables from the source schema where the similarity is above a certain threshold or the $n$ matches with the highest similarity. In the second step, the system then determines suggestions as to which attributes from the corresponding tables fit together. In addition, after the first step, a user can optionally review and select those table pairs that should be kept for the second step.

In the following, we give an overview of the two steps of our matching procedure.

**Table Matching:** Two general approaches are possible here: First, the schema information can be used in the form of table name and attribute names. Tables with semantically similar schema information are probably used to store similar content. Second, exactly these contents of the tables can be examined: we can use the contents of each table to compute embeddings for the tables (either based on all attributes or a subset of it). These embeddings can then be compared to other embeddings to find possible (partial) table matches. Since the candidate pool for this comparison is a cross product of the attributes of all tables which we wanted to avoid with the two-step approach and the calculation of the data embeddings can be expensive to compute, a combined approach is advisable: first the number of table pairs to be examined is reduced schema-based, and then the candidate pool is further reduced using an instance-based approach. For optimization, some

intermediate results of the table matching can be stored for the attribute matching step following afterwards.

**Attribute Matching:** In this step, we only use the table candidates that qualify based on table matching, as discussed before. For the attribute matching, we can again use schema or instance information, analogous to table matching. If no further auxiliary information (e.g., strict type annotations) is available, all attributes of a source table must be compared with those of the target table. In this paper, we are considering again two different types of matchers: Name-based matching uses the table information such as table name and attribute names, as well as available comments. Instance-based matchers inspect the contents of the columns to find attributes with semantically equivalent instances.

## 14.5. Step 1 – Table Matching

In the following, we discuss the details of our first step and present and evaluate different approaches for table matching using schema information and instance information for computing a table embedding.

### 14.5.1. Structure-Based Matching

As discussed before, in a first step we use our table matcher to reduce the search space for the subsequent attribute matching step. It should therefore recognize as many table correspondences as possible (while still delivering as few incorrect correspondences as possible to have a noticeable effect). A high recall is hence more important than good precision.

Table matching is initially a ranking problem (ranking all target tables for a source table based on the supposed similarity). Afterwards, the pool of candidates has to be restricted: by taking the best $n$ correspondences, by applying a similarity threshold $t$, or both. In our experimental evaluation, we therefore check if it is possible to determine suitable values for these parameters.

Gromann et al. [GD18] have shown in their investigations, that for aligning ontologies better results can be achieved with word embeddings than with traditional string comparison methods. We will now investigate whether this also holds for matching relational databases. Furthermore, we examine the use of next-generation embeddings that can dynamically generate vectors for each input, thus avoiding out-of-vocabulary errors and not requiring explicit handling of multi-word expressions.

For computing table similarity using a table embedding that relies on schema information, we represent each table by a vector for the (equally weighted) combination of table name and all attribute names. The representations are determined using the pre-trained Google USE model.

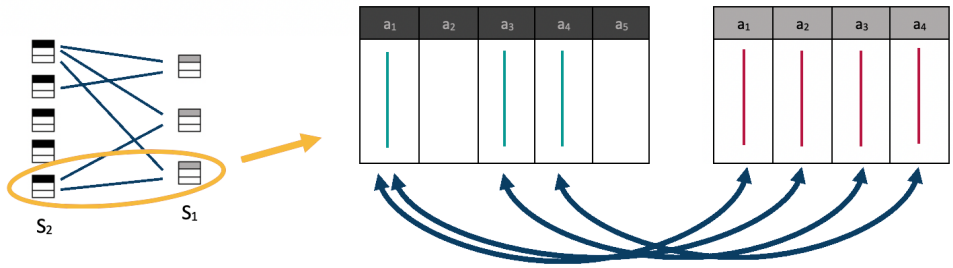



Figure 14.3.: Instance-Based Table Matching – The similarity of tables is inferred from the averaged similarity value for each attribute correspondence.

**Experiment 1: Schema-Based Table Matching:** In this experiment we test the matching exclusively on the basis of schema information. The main goal of this experiment (as well as the other experiments in this section) is to show the matching quality of our approach for table matching in isolation. An end-to-end evaluation and a comparison with existing approaches as baselines will be presented in the experiments in Section 14.6.2.

For the table matching experiments, the English Mondial DB[2] and the German Terra DB[3] are used. Both databases contain geographic content such as mountains, rivers, deserts, or islands and have similar tables since the later one was derived from the first one for teaching purposes [DR13]. We manually aligned them as a gold standard. In the following, we show the results using these data sets with varying parameters.

First, we test the effects of the parameter $n$ (i.e. only the best $n$ correspondences are considered) with a fixed minimum cosine similarity threshold $t$ of $0.5$. The results can be seen in Figure 14.2a. 75% of the correspondences are detected with $n = 5$, with $n > 8$ even 90% or more. Second, we fix $n = 8$ and vary the threshold $t$ (see Figure 14.2b). As expected, the average number of matching candidates shrinks for higher similarity thresholds, and so does the recall while the precision increases. At a threshold of $t = 0.8$ only very few correspondences ($0.2$ per table on average) are suggested, hence the recall is low. However, the remaining suggestions have a good quality, which manifests itself in a high precision ($> 0.55$).

A qualitative analysis of the results shows that the reason for low precision values at lower thresholds is probably due to similar schematic information in the different tables. For example, almost all tables contain the attribute *name* in slightly modified form. Nevertheless, the experiment shows that if the attributes are sufficiently unique, it is possible to determine schema-based table correspondences with embeddings.

### 14.5.2. Instance-Based Matching

Alternatively to only using schema information to compute embeddings, we can also use instances to compute table embeddings. Even with well-existing schema information, additional instance information can help to further increase matching accuracy [Do05]. Studies show that traditional methods can even achieve better results with instance-based matching than with name-based matching (depending on the quality of the instances) [LN07].

[2] `https://www.dbis.informatik.uni-goettingen.de/Mondial/`
[3] `https://www.sachsen.schule/~terra2014/index.php`

In instance-based matching, a distinction is made between horizontal and vertical matching [LN07]. Vertical matchers compare column contents of individual attributes and infer attribute correspondences, while horizontal matchers attempt to identify duplicates (i.e. two or more representations of the same object) between two schemata. There exists work by Ebraheem et al. [Ebr+18] for the recognition of duplicates in schemata (*Entity Resolution* or *Record Linkage*) with word embeddings. They investigate both the use and adaptation of pre-trained models, and the training of new embeddings for this purpose.

We propose an approach for vertical schema matching that uses embeddings as shown in Figure 14.3: embeddings in our approach are used to represent the entire content of a column by a single vector and then to compare them with each other. Since this paper aims at the evaluation of (word) embeddings, we only consider string attributes here. If the resulting vector representations for attributes are sufficiently similar, an attribute correspondence can be assumed.

The most naïve approach to construct this vector is to determine the vector representations of all instances through the pre-trained model, and then to average these to obtain the representation of the attribute. This is inspired by the naïve but solid [ALM16] baseline approach for sentence embeddings, where a sentence is represented by the average of the individual word vectors of all its words.

However, this approach assumes that all instances are equally important. Analogous to stopwords in classical NLP approaches, there may be instances that contribute less to the semantic meaning, for example placeholders like *not-in-universe*, *unknown*, *NONE* etc. Such instances are also called noise [Dil19] and may dominate the representation if no countermeasures are taken. The averaging approach also ignores the order of instances, but these are usually irrelevant to the semantic meaning of an attribute [Geo05].

The aim is therefore to combine instances to form a single vector representation for attributes that most closely reflects the semantics of the attribute, using frequency values but also relativizing them if necessary (to avoid statistical bias [Mug02]). Sampling based on frequency values can be used for this purpose. At the same time, however, the frequency of an instance is not a clear indicator of whether the information is relevant or not: if not applied carefully, sampling might amplify the noise and thus de-emphasize the representation. Three possible sampling methods are:

**Distinct Sampling** Ignore duplicates when generating the combined representation. This might lead to information loss if the instances are not equally distributed.

**N-Random-Sampling** Take $n$ random instances. Each instance is sampled with the same probability, hence to get a random distribution over distinct instances, distinct sampling should be performed first. The subset resulting from the selection is in most cases balanced if the sample sets are large enough so that the column is well represented. The method is considered the safest way to counteract a statistical bias in the resulting subset [Jaw12].

N-Random-Sampling can also be used to validate the semantic representation of a column with word embeddings: for this purpose, at least two sample sets are taken from the column containing randomly selected instances. The comparison of the semantic representations of the two samples should give a cosine similarity close to one. We will use this technique to test our instance representations in Experiment 6.

**N-Most-Common Sampling (with distinct sampling)** Here the instances that occur most frequently are selected. To do this, first a ranking by frequency is created and then each of the $n$ most frequent (distinct) instances is taken to compute the common representation. This method will discard rare values and ensure that even frequent placeholders will only be included once in the representation. Depending on the size of the tables, the calculation can be much more complex than the other two sampling methods. Hence, this method is only beneficial if there are certain instances that occur considerably more frequently than others.

After a representation has been found for the individual attributes, all attributes of the possible source and target table must now be compared to each other. In a basic approach, all possible correspondences with a certain quality are searched for, which requires two threshold values: First, it must be determined from which similarity of the attribute representation one can assume that they match. Second, a ratio is required to determine the minimum fraction of attributes that must match in order for the tables to be considered match candidates.

The choice of this parameter depends on the desired matching scenario: If one only wants to match tables that contain mostly the same types of data, the parameter must be set relatively high. If, on the other hand, tables are also to be found that contain basically the same entities but have different purposes and therefore mostly store different attributes for these entities, it should be close to $0$. In order to be able to distinguish whether two tables really have little in common or whether one table is only more fine-grained than the other, the comparison must also be carried out in both directions. Unless a weighting of the columns is known, all attributes of a table should be considered for that calculation.

An alternative approach to directly comparing individual attribute similarities is to average the similarity values of all pairs of attribute representations to obtain a single similarity value for each pair of tables. These can then be used to derive the best table correspondence(s) for a given table. It can be useful to incorporate not all similarity scores for attribute pairs into this common similarity score for table pairs, but to ignore all attribute pairs with a score below a certain threshold. This allows it to obtain meaningful values for all matching problems where partial matches of tables are also relevant (i.e., not only the best 1:1 correspondence is searched for). Further fine-tuning (e.g., additional normalization with the ratio of selected to overall attributes of a table) may be needed depending on the schemata to match.

**Experiment 2: Instance-Based Table Matching:** In order to see the effect of the parameters of our instance-based table matcher, we again compare the two geographical databases. Although the databases contain many similar instances, a syntactical matcher would find only a fraction of these matches, since one database contains English and the other German content. We use N-Most-Common Sampling and only consider non-numerical attributes. For each attribute in the target table, only the best match in the source table is considered (although it might be selected for multiple target attributes). We require a match of at least half of the attributes ($col\_ratio = 0.5$) and test the effect of different thresholds $t_a$ above which two attributes are assumed to be similar. The results can be found in Figure 14.4. With an instance-based threshold of $t_a = 0.95$, almost 75% of the table correspondences can be found with an average of five table suggestions per target table. A qualitative analysis shows that the representation of columns containing abbreviations or artificial IDs often resemble each other. At this point, an additional syntactical comparison for exact matches could help to distinguish between them.

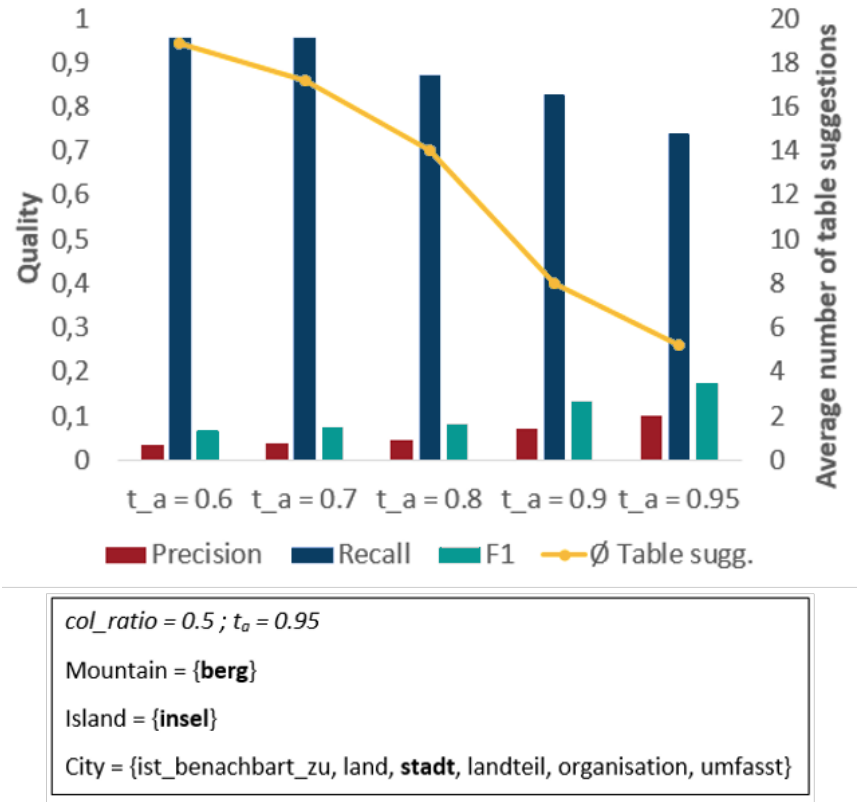
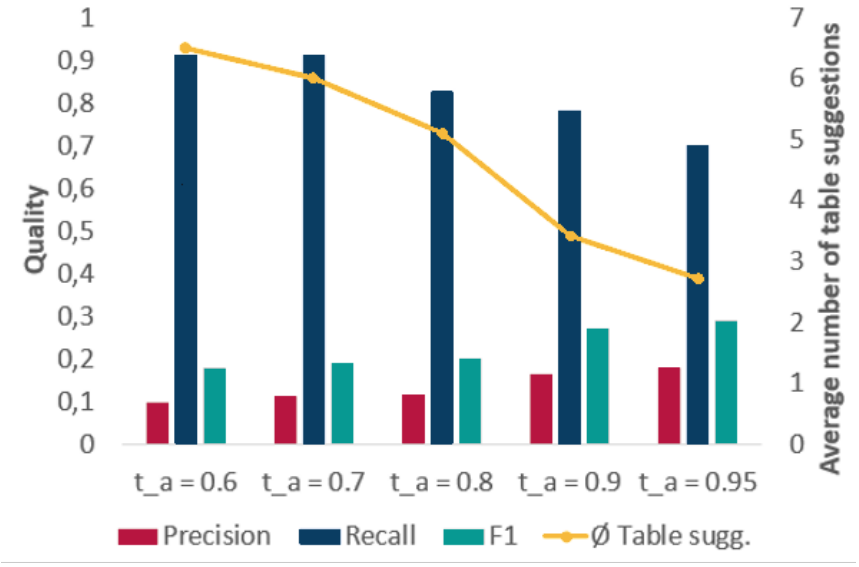



Figure 14.4.: Experiment 2 – Cross-lingual instance-based table matching on geographic databases. Precision, recall and F1 metric as well as the average number of found possible correspondences are reported for different attribute similarity thresholds $t_a$ and a fixed attribute match ratio of 0.5. We also show a qualitative analysis of a fixed parameter setting.

### 14.5.3. Compound Grouping Approach

For the instance-based matching described above, it would be necessary to compare the attributes of all tables with each other, hence the runtime will scale quadratically with the number of attributes and additionally depends on the number of instances. It is therefore advisable to combine the two approaches and work on groups of tables (analogous to the fragment based matching by Do [Do05]), i.e. to first reduce the number of tables to be compared by schema-based matching and then to exclude further candidates on an instance-based basis to increase precision. To obtain a high recall, the thresholds must be selected as low as possible in the first step, whereby the selection should be based on the possible computing effort for the second step, since the computing effort for the first step is negligible in comparison.

**Experiment 3: Matching with Schema-Based Grouping:** In the final experiment for this section, a combined approach will be investigated. For this purpose, all tables with a similarity of at least $t = 0.5$ are determined as described in Experiment 1. These correspondences are then refined instance-based using the procedure described in Experiment 2, again using a $col\_ratio$ of $0.5$. Figure 14.5 summarizes the results for different instance matching thresholds $t_a$. Compared to the




*t = 0.5 ; col_ratio = 0.5 ; $t_a$ = 0.95*

City = {landteil, land, **stadt**}

Continent = {**kontinent**, umfasst}

Country = {landteil, **land,** stadt, umfasst, hat_sitzt_in}

Desert = {**wueste**}

Geo_mountain = {berg, landteil, land, ist_benachbart_zu, **geo_berg**}

Figure 14.5.: Experiment 3 – Two-step cross-lingual table matching on geographic databases. In the first step, schema-based matching with $t = 0.5$ is applied. Afterwards, instance-based matching is performed on the resulting groups. Precision, recall and F1 metric as well as the average number of found possible correspondences are reported for different attribute similarity thresholds $t_a$ and a fixed attribute match ratio of $0.5$. We also show a qualitative analysis of a fixed parameter setting.

experimental results of the individual strategies, the best F1 value was achieved with $t_a = 0.95$. For each target table, an average of $2.7$ tables were suggested, with $70\%$ of the table correspondences being found.

The experiments show that the described procedure can be used to find correspondences that would not be recognized by syntactic matchers without, for example, requiring domain-specific ontologies. With the table matching step, it is possible to severely restrict the set of attributes to be compared, thus to considerably reduce the calculation time for attribute matching or to allow more complex operations to be performed per comparison.

| | *Precision* | *Recall* | *F1* |
|---|---|---|---|
| **COMA NodeNames** | 0.494 | 0.472 | 0.451 |
| **Our Approach (NB)** | 0.502 | **0.516** | **0.507** |
| **COMA AllContextInst** | 0.577 | 0.465 | 0.489 |
| **Our Approach (IB)** | **0.979** | 0.257 | 0.405 |

Table 14.1.: Experiments 4 & 8 – Average values for name-based (NB) and instance-based (IB) version of our approach compared to the two COMA baselines. Values between 0 and 1, higher is better.

## 14.6. Step 2 – Attribute Matching

In order to determine not only tables that might contain related content, but also exact correspondences between attributes of different tables, the candidates from the previous step must now be refined. Depending on the parameter selection, either 1:1 relationships or a list of possible attribute correspondences are created, which can then be used directly or confirmed manually. Table correspondences from table matching for which no attribute correspondence could be confirmed are automatically rejected.

Matching at attribute level is done in a similar way to matching at table level in many places, but the parameters such as weights and thresholds must be selected differently. As before, both structural information (such as table and attribute titles or comments) and the actual instances can be used for this purpose. In the following, the adjustments compared to table matching are explained and the individual approaches are evaluated.

### 14.6.1. Name-Based Attribute Matching

Section 14.5.1 already showed how table and attribute names can be represented and then compared using embeddings. In this section, we use an embedding-based attribute-matcher which only considers the names of the individual attributes without including any additional information like neighboring elements or data type information. The similarity of the attribute-matcher thus purely relies on the cosine-similarity of the embedding representation for two attribute names.

**Experiment 4: Name-Based Attribute Matching:** To compare the attribute-matcher with other matching baselines we use the OAEI Benchmark[4] from 2009, which includes ontologies with different matching challenges. In this benchmark, a reference ontology comprising 33 classes, 64 properties and 76 instances must be matched to modified ontologies. The modified ontologies embody 51 match problems, including, for example, the replacement of element labels with random or foreign words. Since the data to be matched was created by modifying the individual attributes, there are 1:1 matches that must be found.

[4]Ontology Alignment Evaluation Initiative. `http://oaei.ontologymatching.org/2009/benchmarks/`

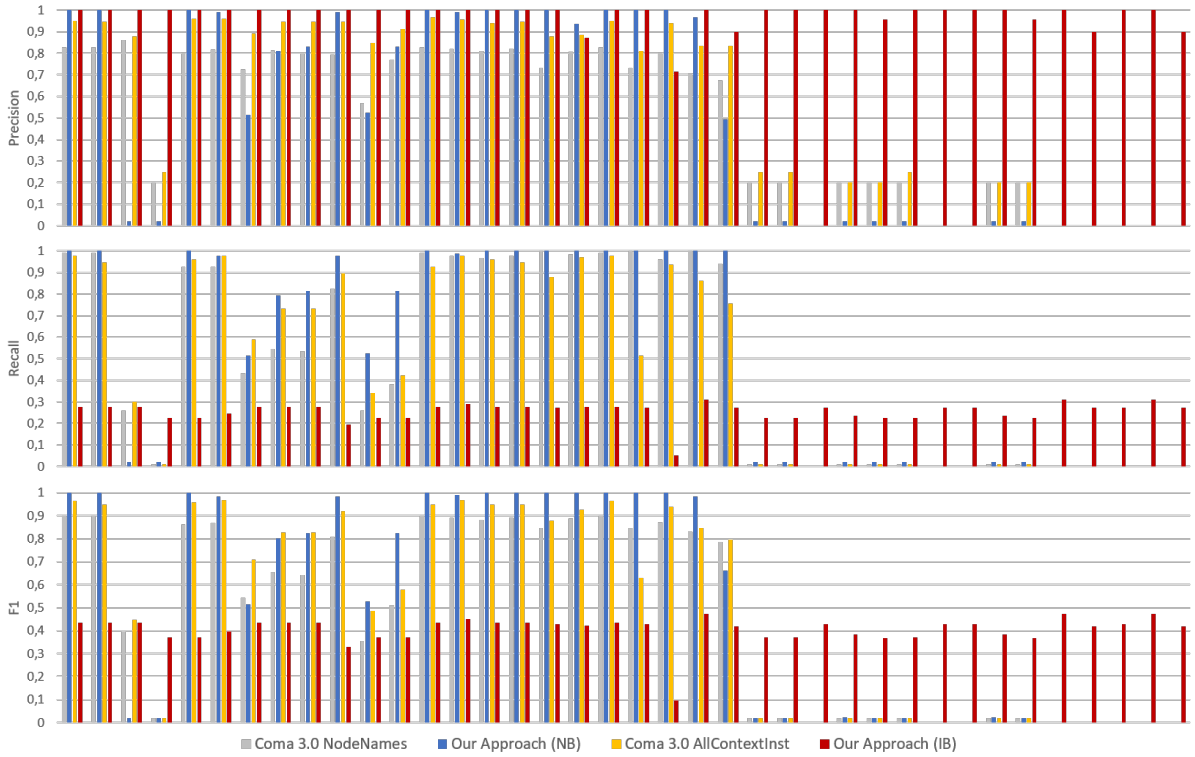



Figure 14.6.: Experiments 4 & 8 – Precision, Recall and F1 Scores for applying the two COMA baselines and the name-based (NB) and instance-based (IB) variant of our approach to all 38 matching problems of the OAEI benchmark.

In order to enable comparability with other strategies, we will consider only the 38 matching problems in which there are instances in general. However, the data is sparse, hence even for them only about 1/3 of the attributes contain instances. As a baseline for comparison we use the COMA 3.0 framework already introduced in Section 14.2, which provides reference implementations for standard matching techniques in so-called workflows. These workflows usually consist of a combination of different (complex) matchers, i.e. the application of similarity metrics (partially using auxiliary information).

For this experiment, we use the `NodesNameW` strategy of COMA 3.0, which only considers the attribute names and evaluates their similarities using syntactical comparison methods such as the Levenshtein distance to show how much a neural embedding matcher can improve over a standard matcher. In later experiments, we also compare to more sophisticated matchers in COMA. Different from the simple COMA matcher, our name-based matcher searches for the best correspondence between vector representations of attribute names. It turns out that the embedding-based approach provides on average both better precision and higher recall (see Table 14.1). The results of the individual matching problems can be found in Figure 14.6. It can be seen that our embedding-based approach almost always performs better for those problems where the name-based variant of COMA already performs well, but performs worse for problems where COMA already has difficulties.

**Experiment 5: Static vs. Contextualized Embeddings:** After having shown that using word embeddings has an advantage over traditional models, we will now examine the intuition mentioned in Section 14.3 that contextualized embeddings are better suited for this purpose than traditional

| | *Aggr.* | *Precision* | *Recall* | *F1* |
|---|---|---|---|---|
| **Our Appr. (W2V)** | CG | **0.65** | 0.26 | 0.37 |
| | Sum | 0.35 | 0.61 | 0.44 |
| **Our Appr. (GloVe)** | CG | 0.52 | 0.32 | 0.4 |
| | Sum | 0.27 | **0.69** | 0.39 |
| **Our Appr. (ELMo)** | - | 0.43 | 0.58 | **0.5** |

Table 14.2.: Experiment 5 - Precision, Recall and F1 for matching on the XDR benchmark using static embeddings (Word2Vec (W2V) & Glove) with Coherent Group (CG) or Sum Aggregation method compared to a contextualized embedding (ELMo). Values between 0 and 1, higher is better.

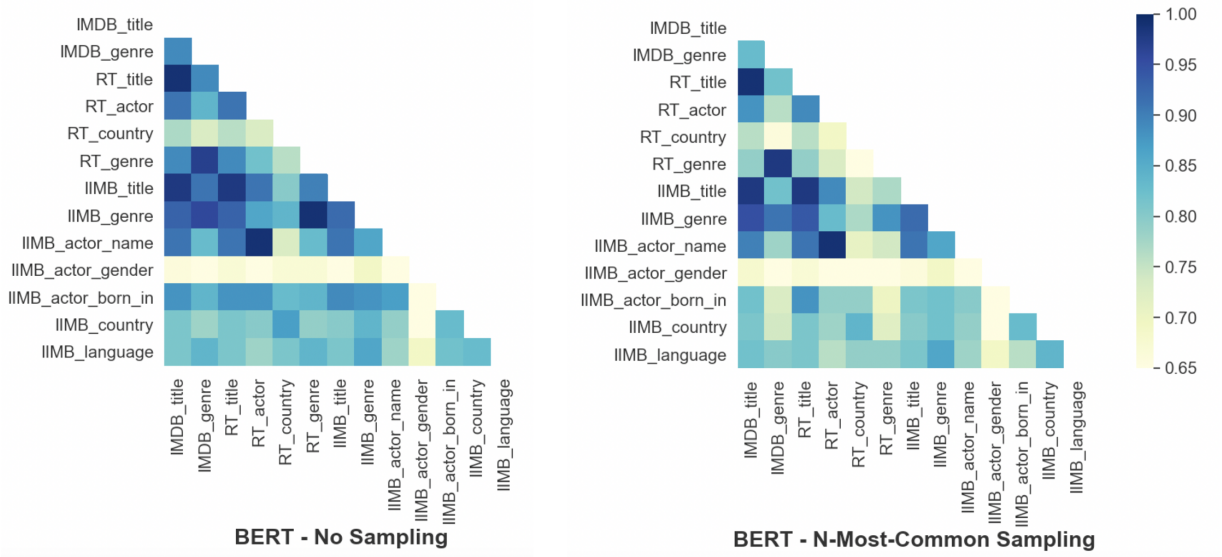



Figure 14.7.: Experiment 7 – Cosine similarities between instance representations for different movie attributes using BERT and different sampling approaches. Values between 0 and 1, similar attributes should have a value close to 1.

static embeddings. The classical static embeddings *word2vec* and *GloVe* are compared with the contextualized *ELMo*. The latter takes the context into account, but in contrast to *BERT* and *Google USE* it is still unidirectional, allowing a fairer comparison.

For this experiment, we use the five XDR schemes[5] for customer orders (CIDX, Excel, Noris, Paragon and Apertum) as a benchmark. They do not contain instances, but only schema elements, and there are many abbreviations and composite element names such as `qty`, `contactName`, `POShipTo`, or `SupplierReferenceNo`. In a pre-processing step, compound words were separated into partial words and known abbreviations such as `PO` were translated into their written form using a dictionary.

The average precision, recall and F1 score can be found in Table 14.2. For each possible schema pair, all attributes were compared using our representation approach with the different embedding models and all attribute pairs with a similarity of at least $0.8$ were selected as correspondence. For the Word2Vec and Glove models, a distinction was made between the two aggregation methods *Coherent Groups* and *Sum* (see Section 14.3). For ELMo the representation could directly be computed using the model itself, regardless of whether the expressions were single or multiple words.

The two static embeddings *word2vec* and *GloVe* reach similar F1 values depending on the aggregation method. While the precision is higher with *Coherent Groups*, more correspondences are found with the *Sum* aggregation method. The contextualized model ELMo is a compromise between Precision and Recall and thus reaches the highest F1 value.

The low recall values when using coherent groups can be explained by OOV errors. For example, in the correspondence (`orderNum, customerOrderRef`) the subwords `num` and `ref` have no vectors and lead to a similarity of 0. Furthermore, for Coherent Groups, the threshold of $0.8$ is very high, since the average value of all partial comparisons of the word groups tends to be lower due to unequal subwords. Therefore, in most cases only attribute correspondences with nearly or complete name equality were suggested.

Sum aggregation, on the other hand, produced many incorrect correspondences, since unequal attributes often contain identical subwords: the attributes `contactName` and `companyName` incorrectly have a similarity of more than $0.8$, since half of the resulting word vectors consist of the vector for the subword `name`.

The contextualized word embeddings model ELMo also frequently produces incorrect correspondences due to identical subwords, but to a lesser extent than the sum aggregation method. Since ELMo can form vectors for whole sentences, these vectors are not only generated purely from the vectors of the subwords, but also from the property of how subwords stand in context to each other. For this reason additional synonyms like (`contactPhone, telephone`) or (`contactEmail, mail`) could be found.

The experiment demonstrates that it is reasonable to not perform the handling of multi-word expressions and domain-specific vocabulary through additional steps, but to use contextualized embeddings that can handle them inherently.

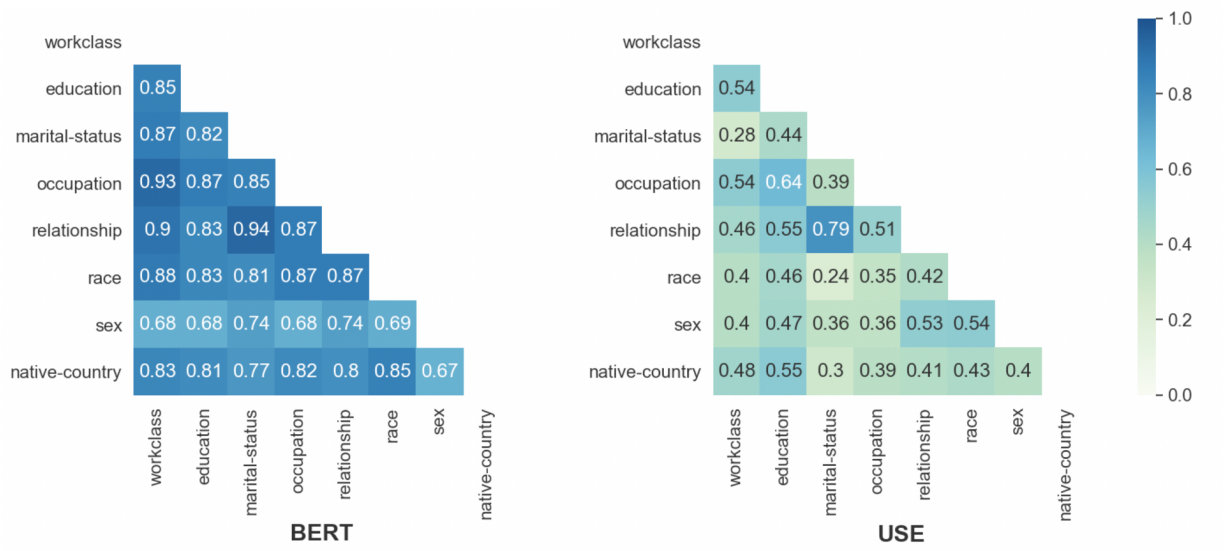



Figure 14.8.: Experiment 9 – Cosine similarities between the instance representations of the attributes of the Adult dataset for representations based on BERT and Google USE. For BERT, dissimilar attributes also have similarity values of at least 0.65, while for Google Use the range is much wider.

### 14.6.2. Instance-Based Attribute Matching

In Section 14.5.2 we introduced the matching of attributes using instances and based on sampling. Again, we use the same embedding for instances of a column as for the table based matcher using either SUM or AVG to compute a vector representation for all (sampled) values of a column. Based on such a column embedding, the cosine distance is used as similarity metric.

**Experiment 6: Robustness of Instance Representations:** For this experiment we again use the OAEI benchmark and additionally two film data sets from the Internet Movie Database (IMDB)[6] and the Rotten Tomatoes Database (RT)[7]. In addition to 6000 to 7500 real film titles, these two data sets contain names of actors, genres and countries. To test the quality of the representation of a column, subsets of the attribute instances are formed according to different patterns, represented and then compared to other representations of the same attribute. We test the following patterns using this data sets:

**Overlapping** Divide Column in two random but equally sized parts (might contain overlaps).

**Distinct** Remove duplicates and then split the remainder into two parts of equal size.

**N-Random** Take $N$ random distinct samples and split those into two parts.

Our results show that for all attributes, the cosine similarity is nearly one for the overlapping method, and at least $0.9$ for the distinct method. Even with $N = 10$, hence only $5$ instance per

[5] https://dbs.uni-leipzig.de/bdschemamatching
[6] https://github.com/AhmedSalahBasha/schema-matching/blob/master/imdb.csv
[7] https://github.com/AhmedSalahBasha/schema-matching/blob/master/rotten_tomatoes.csv

vector, the representations have a similarity of at least $0.8$ (e.g., for IMDB genres or surnames). If the instances have some internal structure (for example, non-normalized addresses consisting of streets and house numbers), the similarity measures are even closer to $1$ even for these very small samples.

However, it is not sufficient that the representations are robust even with sampling, they must also differ as much as possible from the representations of the other attributes. Therefore, in the following we will further investigate the effect of sampling.

**Experiment 7: Effects of Instance Sampling:** For this experiment we again use the two movie data sets and additionally the movie-specific IIMB ontology from the OAEI benchmark. For all attributes, representations without sampling and with Distinct N-Most-Common sampling (see Section 14.5.2) are calculated. The sample size is adapted to the attribute in a way that it takes into account all instances appearing more frequently than average (considering the standard deviation). The results using the widely used BERT embeddings can be found in Figure 14.7. For attributes with a very high degree of similarity, it usually remains high even after sampling, especially for attributes that are actually semantically similar (e.g., `IMDB_title` and `RT_title`). Faulty matches are partially attenuated by sampling, but there are cases where the similarity remains high (e.g., `IIMB_actor_born_in` and `RT_title`). Generally it is found that sampling is useful for spreading the range of values and thus ensuring that dissimilar attributes have a lower similarity value while the similarity value of related attributes remains high.

Now that we have examined the components of the instance-based approach, we will evaluate the quality of the actual matching and compare it to other approaches, both traditional and embedding-based.

**Experiment 8: Instance-Based Attribute Matching:** Analogous to Experiment 4, we again use the OAEI benchmark and compare our approach with a COMA reference implementation. As baseline, we use the more complex COMA 3.0 matching workflow `AllContextInstW`. Among other things, it analyzes schema information such as table paths and element names, but also includes the instances themselves and, according to the authors, should (only) be used if instance information is available. It is meant to be used stand-alone. In order to avoid introducing additional parameters, we compare it to an embedding-based matcher that works purely on instances. For productive use, however, it would make sense to combine this approach with a name-based approach and the correspondence pre-selection step.

The average results can again be found in Table 14.1, the results for the individual matching problems can be found in Figure 14.6. Our approach provides very high precision. The seemingly low recall can be explained by the fact that there are instances for only about 1/3 of the attributes. Most of the correspondences of these attributes can be found by the instance-based matcher, which can roughly maintain its quality-level even for the problems where the other three approaches have severe difficulties. Instance-based matching can therefore help to find correspondences when attribute names are not meaningful or semantically difficult to compare, for example due to domain-specific language. Overall, one can assume that a balanced combination of our two approaches would beat both the simpler and the complex (combined) baseline approach.

### 14.6.3. How useful is Google USE?

For the evaluation of our methods in this paper we mainly used vector representations based on Google USE (see Section 14.3.2 for the advantages). In the following we will briefly draw a comparison to the well-known BERT embeddings.

**Experiment 9: Quality of Word Embedding Model:** For this we proceed analogously to Nozaki et al. [NHN19], who investigated instance-based matching with the classical Word Embedding model *Word2Vec*: For all attributes an averaged representation is calculated as described in Section 14.5.2 without sampling—once with BERT and once with Google USE. This representation is then compared with the representation of all other attributes.

The results for this experiment on the Adult dataset,[8] which Nozaki et al. used in their experiments too, can be found in Figure 14.8. In both models, the similarity metric is highest between the attributes marital-status and relationship, which correspond semantically. However, with BERT, even very different attributes have similarity metric values of at least $0.65$, while the range is significantly larger when using Google USE. This shows that at least without any special fine-tuning the use of Google USE is preferable for the purpose of attribute matching.

### 14.6.4. Runtime

As a reference for the determination of the computation time for semantic representations with Google USE, we measured the runtimes for the RT dataset on a server with NVIDIA® TESLA® V100 graphics card with 16GB memory. The implementation of the instance-based and thus most complex matching took under three minutes to calculate all column vectors of the RT dataset (which has 10 columns with an average of $10,000$ instances).

## 14.7. Conclusion & Future Work

We have shown that neural word embeddings can be utilized to propose a small set of possible candidates for schema matching which is crucial for data integration. Our experiments prove that word embeddings can be used to bridge the semantic gap for several matching variants. Our approach can be used instead of but in particular as a supplement to existing syntactic and semantic matchers.

The use of models pre-trained for general tasks seems to be sufficient as long as the database does not predominantly contain abbreviations and very specific terms. It is advisable to use contextualized embeddings.

Both structural data like schema information and comments as well as the textual data instances themselves can be used for matching, whereby the effectiveness of the individual approaches strongly depends on the schemata. In instance-based approaches, sampling can help to increase the distinction between similar and dissimilar attributes while reducing the number of instances to be considered. This also makes it possible to use the approach for databases with very large numbers

[8] `http://archive.ics.uci.edu/ml/datasets.html`

of instances. Instance-based approaches work well with different types of entities. Weaknesses are found, for example, with attributes that all contain human names: the embeddings are good for finding other attributes with names, but a further subdivision (e.g., by actors and directors) is difficult.

In the future, it would be interesting to further look at pre-processing of the textual elements before they are represented by embeddings (e.g., resolving of acronyms) and to include additional auxiliary information such as thesauri. It should also be checked whether the results can be further improved by fine-tuning of the embeddings: on the one hand, this could be done by additional training on corpora with domain-specific texts, on the other hand, analogous to Kolyvakis et al. [KKK18], one could try to adapt the word vectors to the existing schema matching problem by using structured external domain knowledge. As an orthogonal problem, the instance-based approach could be further improved to additionally support purely numerical attributes, i.e., to examine if it is possible to learn some kind of embedding for sets of numerical values beyond existing approaches like regular expressions, value range limits, averages, etc. Finally, one could also further investigate matchers that use embeddings to compare natural language comments, which are available for some schemata, with each other or with table and attribute names.

# 15. WannaDB: Ad-hoc Structured Exploration of Text Collections Using Queries (DESIRES'21)

> Just tell me what you want, what you really, really want!

## Bibliographical Information

The content of this chapter was previously published in the peer-reviewed work "Benjamin Hättasch. 'WannaDB: Ad-hoc Structured Exploration of Text Collections Using Queries'. In: *Proceedings of the Second International Conference on Design of Experimental Search & Information REtrieval Systems, Padova, Italy, September 15-18, 2021*. Volume 2950. CEUR Workshop Proceedings. CEUR-WS.org, 2021. URL: https://ceur-ws.org/Vol-2950/paper-23.pdf". The contributions of the author of this dissertation are summarized in Section 5.2.



## 15.1. Motivation

In many domains, users face the problem of needing to quickly extract insights from large collections of textual documents. For example, imagine a journalist who wants to write an article about airline security that was triggered by some recent incidents of a well-known US airline. For this reason, the journalist might decide to explore a collection of textual accident reports from the *National Transportation Safety Board* in order to answer questions like 'What incident types are the most frequent ones?' or 'Which airlines are involved most often in incidents?'. And clearly, there are many more domains where end users want to explore textual document collections in a similar fashion.

Yet, existing approaches to answer such queries over new text collections force users to either read through vast amounts of text and manually extract the relevant information before they can compute an answer to their query or to build extraction pipelines (e.g., when using [Sa+16]) which however require substantial efforts.

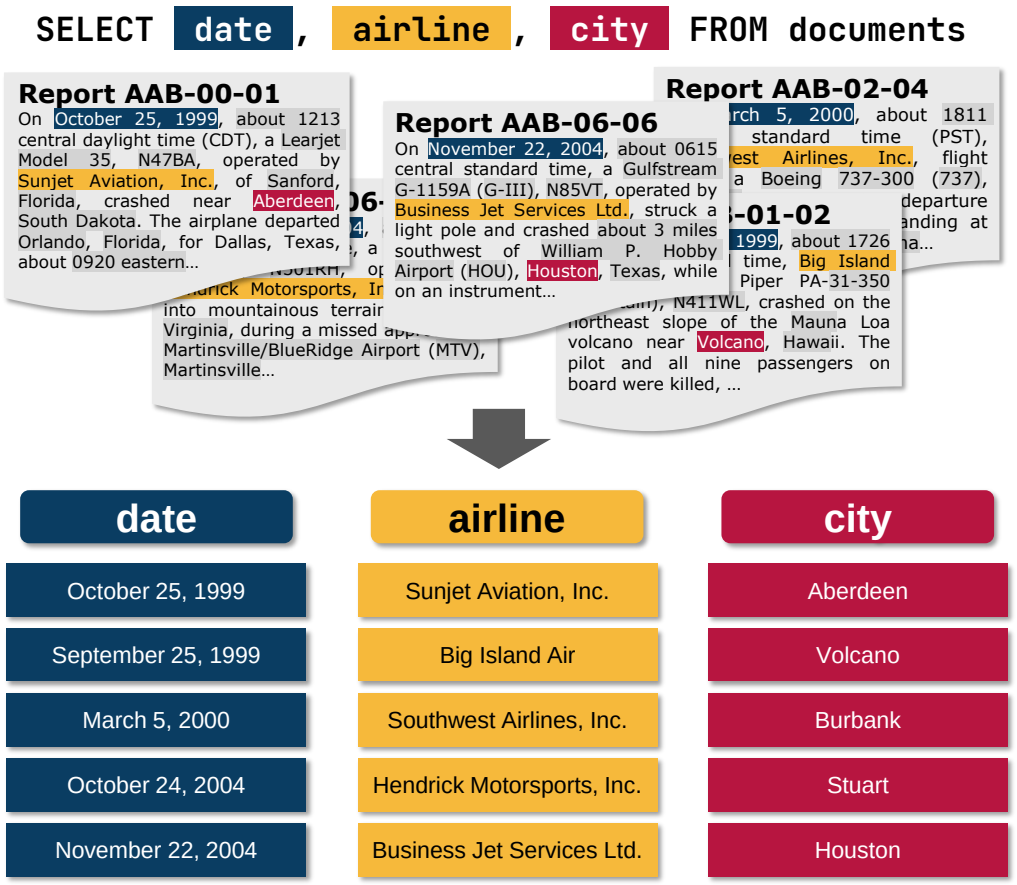



Figure 15.1.: Aim: Query a text collection and receive an approximate structured result without manual extraction

Hence, we advocate for a different route where users can extract structured data relevant to satisfy an information need from a collection of text documents without the need to program, train or specify extraction systems.

Instead, the aim is to provide a system that allows users to explore new (unseen) text collections by simply issuing a query to receive structured information from the corpus. In contrast to [ZB18] this should not require data already in tabular form, rather the idea is to automatically identify the relevant target structure and then, again automatically, fill it from unstructured text.

## 15.2. Contributions

Therefore, we propose WannaDB, a system for ad-hoc structured text exploration. The main idea of WannaDB is that a user specifies their information need by composing SQL-style queries over the text collection. For example, in Figure 15.1, the user issues a query to extract information about dates, airlines, and cities of incidents. WannaDB then takes the query and evaluates it over the given document collection by automatically populating the table(s) required to answer the query with information nuggets from the documents.

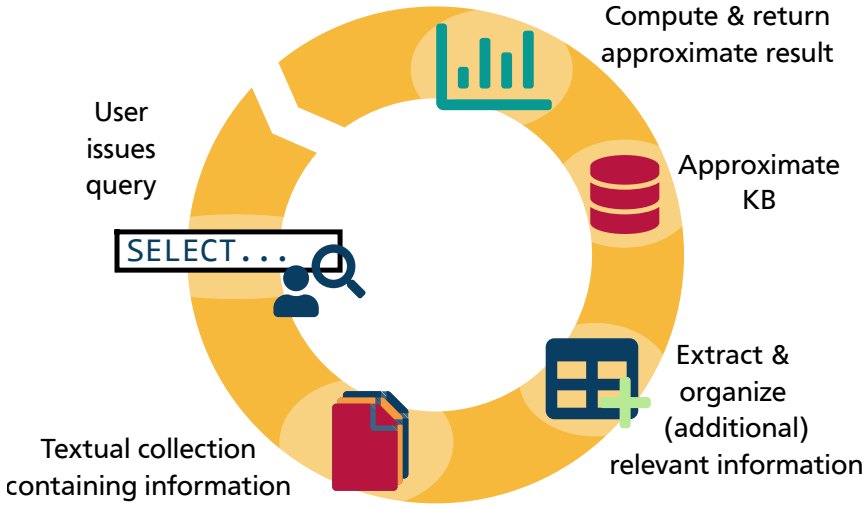



Figure 15.2.: Pipeline & Usage of WannaDB

To do this, WannaDB uses a novel pipeline as shown in Figure 15.2 which first extracts a superset of information nuggets from texts (e.g., all named entities), then determines the information need from the query, and finally matches nuggets to the relevant attributes of the user's query. As a result, WannaDB allows answering the queries, even if the information is not explicitly stated in the corpus but has to be calculated (e.g., when the query contains aggregation functions like `AVG` or `SUM`). A main observation here is that in many cases a sample of extractions (i.e., a table with partially missing or incorrect values) is sufficient to produce approximate results to answer the user's query.

## 15.3. Architecture & Initial Evaluation

WannaDB contains components to determine the information need from queries, aggregate the relevant information, and compute the actual query result. For the extraction of possibly relevant information nuggets, WannaDB relies on off-the-shelf extractors like Stanza [Qi+20]. The key contribution of WannaDB, however, is a new matching approach that uses a novel embedding space exploration algorithm incorporating interactive user feedback: The matching process is done separately for each relevant attribute. It starts by selecting information embeddings close to the attribute embedding. Afterwards, other embeddings that might be matches are searched by applying several selection rules based on the closeness of embeddings to known matches. Each candidate is presented for feedback (yes/no) to the user. The algorithm balances between exploration and exploitation to select those information nuggets for feedback that quickly allow identifying the areas in the embedding space relevant for the attribute with as little feedback as possible. This area can then be used to populate the remaining rows in the target table. Previously extracted information (and user feedback) can be reused for follow-up queries. A detailed description of the matching process can be found in [HBB21].[1]

Our experiments on different text-collections each focused around certain topics lead to promising results: $10 - 25$ quick iterations of feedback for each attribute (i.e., confirming whether an

[1] See next Chapter

information nuggets belongs in a certain column of a table) sufficed for high matching scores for both textual and numeric attributes.

## 15.4. The Road Ahead

In the next steps, we want to enlarge the scope, i.e., support more general corpora. We also want to leverage certainties from the extraction and matching process for the computation of the approximate result and provide useful interfaces for the end users, not only through a standalone application but also e.g., in form of a Jupyter Notebook extension.

# 16. *ASET*: Ad-hoc Structured Exploration of Text Collections (AIDB'21)

## Abstract

In this paper, we propose a new system called *ASET* that allows users to perform structured explorations of text collections in an ad-hoc manner. The main idea of *ASET* is to use a new two-phase approach that first extracts a superset of information nuggets from the texts using existing extractors such as named entity recognizers and then matches the extractions to a structured table definition as requested by the user based on embeddings. In our evaluation, we show that *ASET* is thus able to extract structured data from real-world text collections in high quality without the need to design extraction pipelines upfront.

## Bibliographical Information

The content of this chapter was previously published in the peer-reviewed work "Benjamin Hättasch, Jan-Micha Bodensohn, and Carsten Binnig. 'ASET: Ad-hoc Structured Exploration of Text Collections'. en. In: *3rd International Workshop on Applied AI for Database Systems and Applications (AIDB21). In conjunction with the 47th International Conference on Very Large Data Bases, Copenhagen, Denmark, August 16 - 20, 2021*. Copenhagen, Denmark, 2021. arXiv: 2203.04663". The contributions of the author of this dissertation are summarized in Section 5.2.

## 16.1. Introduction

**Motivation:** In many domains, users face the problem of needing to quickly extract insights from large collections of textual documents. For example, imagine a journalist who wants to write an article about airline security that was triggered by some recent incidents of a well-known US airline. For this reason, the journalist might decide to explore a collection of textual accident reports from the *National Transportation Safety Board* in order to answer questions like 'What incident types are the most frequent ones?' or 'Which airlines are involved most often in incidents?'. To be able to formulate such answers to their questions, they would need to extract the relevant information, then create a structured data set (e.g., by creating a table in a database or simply by using an Excel sheet) and analyze frequency statistics such as the number of incidents per airline.

And clearly, there are many more domains where end users want to explore textual document collections in a similar fashion. As another example, think of medical doctors who want to compare symptoms and reactions to medical treatments for different groups of patients (e.g., old vs. young, w/ or w/o a specific pre-existing condition) based on the available data coming from textual patient reports. To do this, the doctor again would need to extract the relevant structured information about age, pre-diseases, etc. from those reports before being able to draw any conclusions.

One could now argue that extracting structured data from text is a classical problem that various communities have already tackled and several industry-scale systems already exist. For example, DeepDive [Sa+16] or System-T [Lem+20] are examples of such systems that have developed rather versatile tool suites to extract structured facts from textual sources. However, these systems typically require a team of highly-skilled engineers that curate extraction pipelines to populate a structured database from the given text collection or train machine learning-based extraction models that come with the additional need to curate labeled training data. A major problem of these solutions is the high effort they require and, thus, it can take days or weeks to curate such extraction pipelines even if experts are involved. Even more importantly, such extraction pipelines are typically rather static and can extract only a pre-defined (i.e., fixed) set of attributes for a certain text collection only. This typically prevents more exploratory scenarios in which users ask ad-hoc queries where it is not known upfront which information needs to be extracted or whether a new data set should be supported on-the-fly.

**Contributions:** In this paper, we thus propose a new system called *ASET* that allows users to explore new (unseen) text collections by deriving structured data in an ad-hoc manner; i.e., without the need to curate extraction pipelines for this collection.

As shown in Figure 16.1, the main idea is that a user specifies their information need by composing SQL-style queries over the text collection. For example, in Figure 16.1, the user issues a query to extract information about dates, airlines, and cities of incidents. *ASET* then takes the query and evaluates it over the given document collection by automatically populating the table(s) required to answer the query with information nuggets from the documents. An important aspect here is that using *ASET*, users can define their information needs by such a query in an ad-hoc manner.

*ASET* supports this ad-hoc extraction of structured information by implementing a new two-phase approach: In the *extraction phase*, a superset of information nuggets is extracted from a text collection. Afterwards, the information nuggets are matched to the required attributes in the *matching phase*. To do so, *ASET* implements a new interactive approach for matching based on

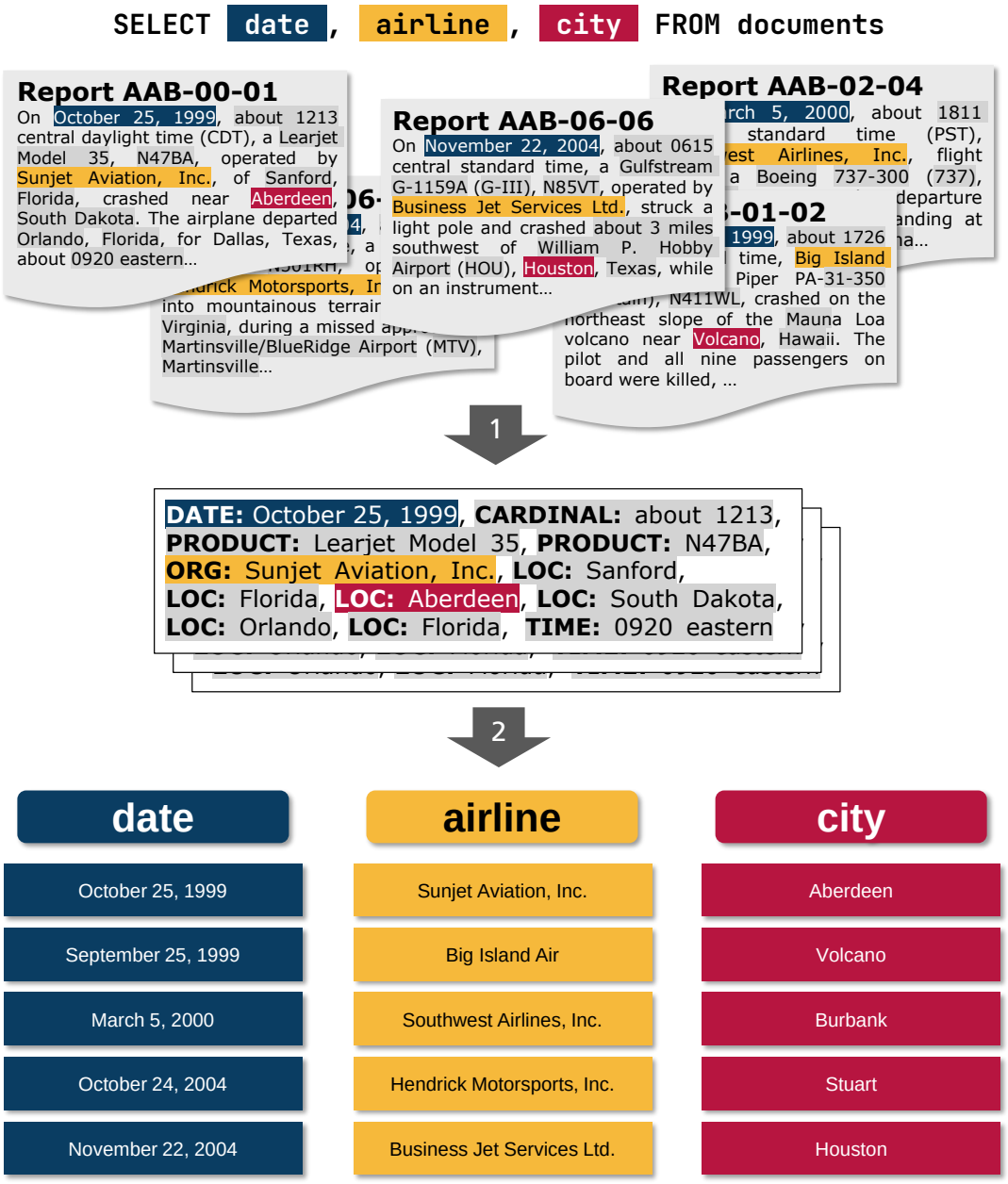



Figure 16.1.: Ad-hoc structured exploration of text collections with *ASET*: (1) a superset of information nuggets is first extracted from the texts and then (2) matched to the relevant attributes of the user query.

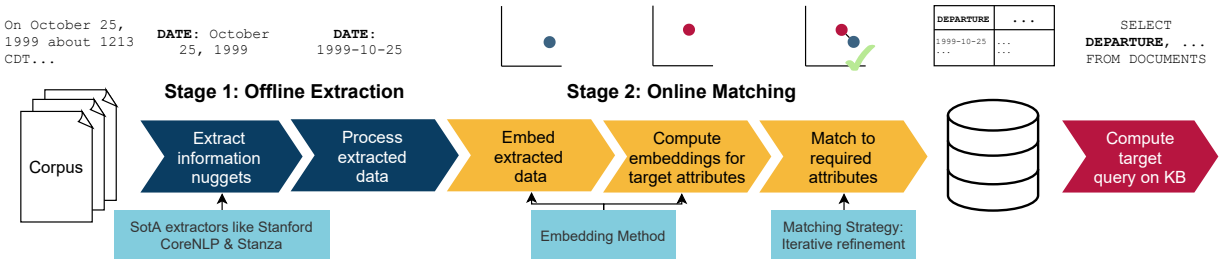



Figure 16.2.: Architecture of *ASET*: The extraction stage obtains information nuggets from the documents. The matching stage matches between the extracted information nuggets and the user's schema imposed by their query.

neural embeddings, which uses a tree-based exploration technique to identify potential matches for each attribute.

Clearly, doing the matching for arbitrarily complex document collections and user queries is a challenging task. Hence, in this paper, we aim to show a first feasibility study of our approach. To that end, we focus on so-called *topic-focused* document collections here. In these collections every document provides the same type of information (e.g., an airline incident) meaning that each document can be mapped to one row of a single extracted table. Note that this is still a challenging task, since arbitrary information nuggets must be mapped to an extracted table in an ad-hoc manner. Clearly, extending this to more general document collections and queries that involve multiple tables is an interesting avenue of future work.

To summarize, as the main contribution in this paper, we present the initial results of *ASET*. This comprises a description of our approach in Sections 16.2 and 16.3 as well as an initial evaluation in Section 16.4 on two real-world data sets. In addition, we provide code and data sets for download[1] along with a short video of *ASET*.[2]

## 16.2. Overview of our Approach

Figure 16.2 shows the architecture of *ASET*. As mentioned before, *ASET* comprises two stages: (1) the first stage extracts a superset of potential information nuggets from a collection of input documents using extractors. This step is independent of the user queries and can thus be executed offline to prepare the text collection for ad-hoc exploration by the user. (2) At runtime, a user issues several queries against *ASET*. To answer a query, an online matching stage is executed that aims to map the information nuggets extracted in the first stage to the attributes of the user table as requested by a query. We make use of existing state-of-the-art approaches for information extraction in the first stage. The core contribution of *ASET* is in the matching stage which implements a novel tree-based approach as we discuss below.

[1] `https://link.tuda.systems/aset`
[2] `https://link.tuda.systems/aset-video`

### 16.2.1. Stage 1: Offline Extraction

As shown in Figure 16.2, the extraction stage is composed of two steps. First, it derives the information nuggets as label-mention-pairs (e.g., a date and its textual representation) from the source documents using state-of-the-art extractors. Afterwards, a preprocessing step is applied, which then canonicalizes the extracted values.

**Extracting Information Nuggets:** The extractors process the collection document by document to generate the corresponding extractions. Clearly, a limiting factor of *ASET* is which kinds of information nuggets can be extracted in the extraction stage since only this information can be used for the subsequent matching stage. For this paper, we successfully employed state-of-the-art information extraction systems, focusing particularly on named entity recognizers from *Stanford CoreNLP* [Man+14] and *Stanza* [Qi+20]. In general, *ASET* can be used with any extractor that produces label-mention pairs; i.e. a textual mention of a type in the text (e.g., *American Airlines*) together with a label representing its semantic type (e.g., *Company*). Moreover, additional information about the extraction (e.g., the full sentence around the mention) will also be stored and used for computing the embeddings, as we describe below.

**Preprocessing Extracted Data:** In the last step of the extraction stage, the extractions are preprocessed to derive their actual data values from their mentions (i.e. a canonical representation). For this we also rely on state-of-the-art systems for normalization: As an example, we employ Stanford CoreNLP's [Man+14] built-in, rule-based temporal expression recognizer SUTime for normalization of dates (e.g., turn *October 25, 1999* and *25.10.1999* into *1999-10-25*).

### 16.2.2. Stage 2: Online Matching

The second stage must match the extracted information nuggets to the user table to answer the query. This stage consists of computing embeddings for the information nuggets and matching them to the target attributes, using a new tree-based technique to identify groups of similar objects in the joint embedding space of attributes and information nuggets.

**Computing Embeddings:** A classical approach to compute a mapping between information nuggets and attributes of the user table would be to train a machine learning model in a supervised fashion. However, this would require both training time and a substantial set of labeled training data for each attribute and domain. Instead, our approach leverages embeddings to quantify the intuitive semantic closeness between information nuggets and the attributes of the user table.[3] For the attributes of the user table, only the attribute names are available to derive an embedding. To embed the information nuggets extracted in the first stage, however, we can use more information and incorporate the following signals from the extraction: (1) *label* – the entity type determined by the information extractor (e.g. *Company*),[4] (2) *mention* – the textual representation of the entity in

[3] We use Sentence-BERT [RG19a] and FastText [Mik+18] to compute embeddings for the natural language signals.

[4] We map the named entity recognizers' labels like *ORG* to suitable natural language expressions according to the descriptions in their specification.

the text (e.g., *US Airways*), (3) *context* – the sentence in which the mention appears, (4) *position* – the position of the mention in the document.

**Matching Step:** To populate the values of a row (i.e. to decide whether an extraction is a match for an attribute of a user table, like a specific DATE instance matches DEPARTURE in Figure 16.2), we use a new tree-based technique to identify groups of related information nuggets that map to a user attribute as we discuss in the next section. Using this technique, we suggest potential matches to the user, who can confirm or reject those matches in an interactive manner (i.e., by reading the extracted value and its context sentence). Previous approaches use only a distance metric (e.g., cosine distance) and often suffer from the curse of dimensionality, not providing a robust similarity metric in higher-dimensional embedding spaces. Our approach allows users to quickly explore the embedding space and find matches between extracted information nuggets and attributes more efficiently.

## 16.3. Interactive Matching

In this section, we first give an overview of the interactive matching process before we discuss the details of how *ASET* selects potential matches to present to the user. The matching is done individually for the different attributes.

### 16.3.1. Overall Procedure

*ASET* implements an interactive matching procedure by confronting the user with information nuggets derived from the document collection and asking them whether those nuggets belong to a particular attribute. The main goal of the interactive matching procedure is to identify groups of information nuggets in the embedding space belonging to a particular user-requested attribute (e.g., airline names or incident types) as quickly as possible (i.e. after a low number of interactions). However, finding information nuggets to present to the user as potential matches is not trivial. Clearly, a first naive idea is to choose extractions that are close to the embedding of the requested attribute (e.g., airline name) using a distance metric (e.g., cosine similarity). Yet, matches identified by these simple distances provide only a limited information gain for identifying additional groups of related objects in a high-dimensional space. Therefore, we are using a new tree-based exploration strategy that can efficiently identify potential matches for each user-requested attribute as we discuss next.

### 16.3.2. Tree-based Exploration Strategy

To identify the group of information nuggets in the embedding space that belong to a user-requested attribute, we use the notion of a tree of confirmed matching extractions (instead of a set as often used by kNN-based approaches). Different from kNN-searches, subspace clustering, or other techniques tackling similar problems, using a tree-based representation allows us to implement a new *explore-away* strategy that can grow the covered embedding space for the group of related

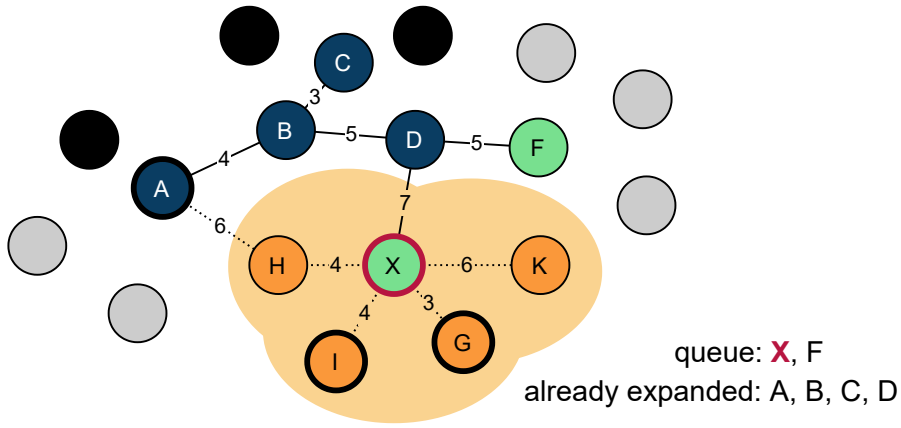



Figure 16.3.: Sketch of the tree-based (explore-away) strategy (executed per attribute). Each node represents an embedded information nugget to be included in the search tree. Confirmed matches are marked in ■ blue, the next candidates in ■ green. Rejected nuggets are marked in ■ black, unexplored ones in ■ gray. Node X is selected for expansion, the nodes closest to X are marked in ■ orange. The candidates selected by our explore-away strategy for user-feedback are G and I.

information nuggets with every confirmed match. Our tree-based exploration strategy works in three steps:

**1. Find a root node::** First, the exploration strategy finds an initial matching node to serve as the root of the tree. This is done by sampling extractions based on their distance to the initial attribute embedding (based only on the attribute name). We start with low distances that result in conservative samples close to the initial attribute embedding, and gradually raise the sampling temperature to include samples from farther away if the close-by samples do not yield any matching extractions to select as root.

**2. Explore-away Expansion::** As a second step, we now explore the embedding space by expanding the search tree using our explore-away strategy in the embedding space. We explain the expansion step based on the example in Figure 16.3 where node $X$ is to be expanded. To expand the node $X$, we determine its potential successors $\mathsf{succ}(X)$ based on the following two constraints: (1) The extractions in $\mathsf{succ}(X)$ must be closer to $X$ than to any other already expanded extraction (e.g., nodes $G$, $H$, $I$, and $K$ qualify in our example). (2) The extractions in $\mathsf{succ}(X)$ must be farther away from the rest of the tree than the node we expand (e.g., $H$ is closer to $A$ than $X$ is to its parent (and hence closest node) $D$ and therefore not a candidate; however, nodes $G$, $I$, and $K$ remain as candidates).

Afterwards, the search strategy selects the $k$ nuggets[5] in $\mathsf{succ}(X)$ that are closest to $X$ (e.g., $G$ and $I$ in our example) to gather user feedback. A user can then confirm whether the proposed nuggets actually match the attribute; matching nuggets are added to a queue of nuggets to be expanded in the next iterations. In case the queue is empty, the explore-away strategy returns to step 1 to start

[5]This number determines the degree of the search trees. We experimented with different degrees and found that 2 results in the best performance.

with an additional root node, or it terminates if a user-defined threshold of confirmed matches is reached. Overall, this procedure thus identifies groups of information nuggets (represented as trees) that match to a certain user-requested attribute.

**3. Static Matching::** Once a user-defined threshold of confirmed matches is reached for every user attribute, *ASET* stops collecting feedback and continues with a static matching procedure: *ASET* leverages the distances between the embeddings of extracted information nuggets and the different groups of embeddings identified in step 2 to populate the remaining cells without asking the user for feedback.

## 16.4. Initial Evaluation

In this section, we present an initial evaluation of our approach, showing that *ASET* can provide high accuracies. For the evaluation, we use two different data sets, which we provide for download together with our source code. We also have additional results showing that our novel tree-based exploration technique is superior over using other techniques that are only based on distance metrics (e.g., cosine similarity) in the embedding space. However, due to space restrictions, we could not include these results in this paper.

**Data Sets:** We perform our evaluation on the two real-world data sets: *Aviation* and *COVID19RKI*. Both data sets consist of a document collection and structured tables to serve as ground truth.

The *Aviation* data set is based on the executive summaries of the Aviation Accident Reports published by the United States National Transportation Safety Board (NTSB).[6] Each report describes an aviation accident and provides details like the prevailing circumstances, probable causes, conclusions, and recommendations. As a ground-truth, we compiled a list of 12 typical attributes and manually created annotations that capture where the summaries mention the attributes' values.

The second data set is based on the German RKI's daily reports outlining the current situation of the Covid-19 pandemic (e.g., laboratory-confirmed Covid-19 cases or the number of patients in intensive care) in Germany.[7] The focus of this data set is to evaluate whether our system can also cope with numerical values which are particularly challenging for matching. To that end, we compiled a list of seven numeric attributes and manually annotated the occurrences of those attributes in the data set.

**Initial Results:** In this initial experiment, we evaluate the end-to-end performance of our system. As a baseline to compare the quality of the matching stage of *ASET* to, we use COMA 3.0 [DR02] which implements a wide set of classical matching strategies that also work out-of-the-box (i.e., similar to *ASET* they do not need to be trained for every new attribute to be matched).

Figure 16.4 shows the result of running *ASET* (using *Stanza* in the extraction stage) with 25 interactions per attribute in the matching stage. We report the average F1-score as well as the

[6] `https://www.ntsb.gov/investigations/AccidentReports/Pages/aviation.aspx`
[7] `https://www.rki.de/DE/Content/InfAZ/N/Neuartiges_Coronavirus/Situationsberichte/Gesamt.html`

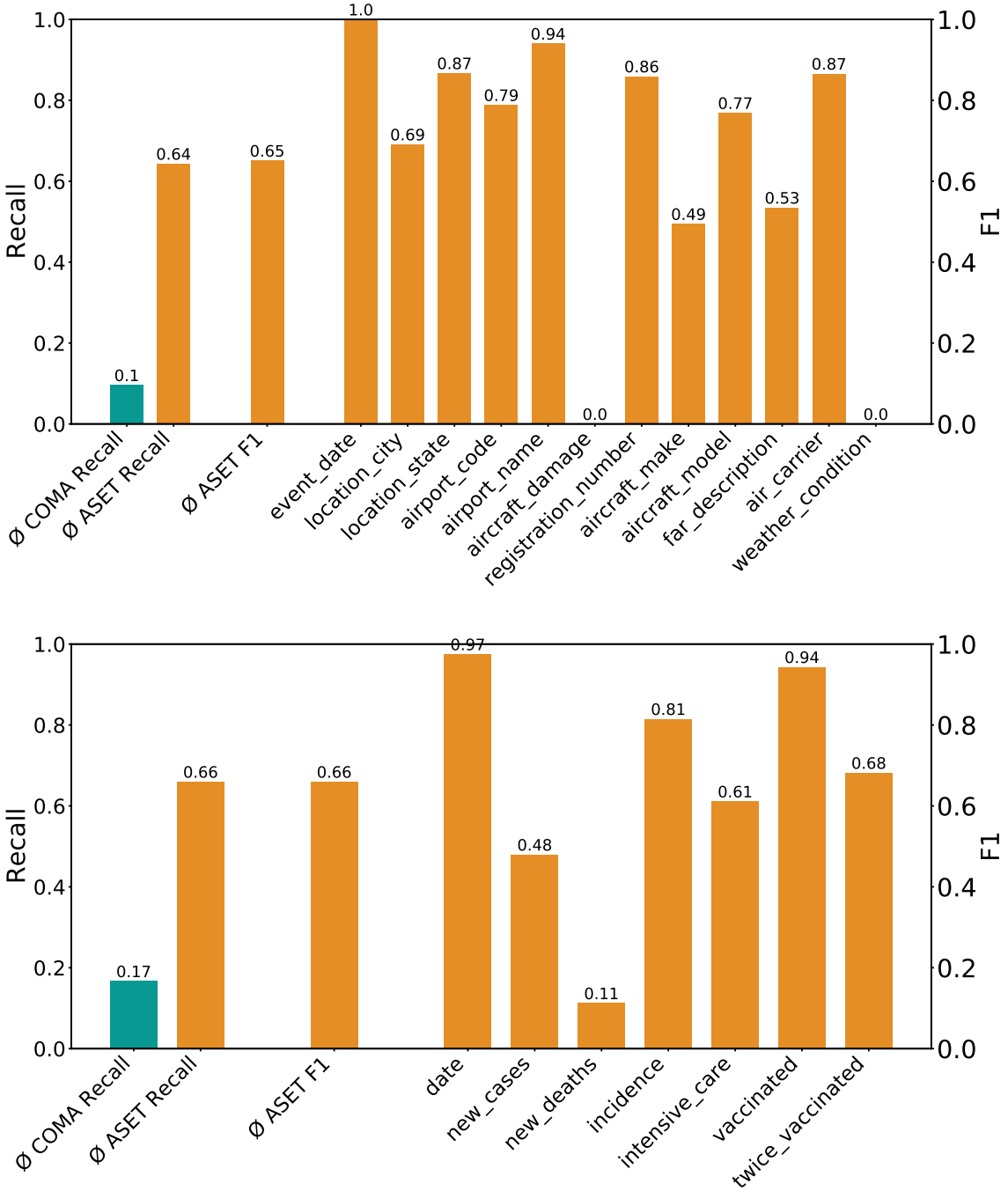



Figure 16.4.: End-to-end evaluation (both stages) of our system on both the *Aviation* and the *COVID19RKI* data set. We report the avg. F1-score and the F1-scores for all attributes for *ASET* (all scores ranging from 0 to 1, higher is better).

individual F1-scores for all attributes. The results show that *ASET* is able to accurately match the extractions for most of the attributes of both data sets. For COMA 3.0, we decided to report only the recall since the precision of matching values was overall low (depending on the selected workflows it either finds hardly any matches or thousands of wrong matches).

## 16.5. Conclusions and Future Work

In this paper, we have proposed a new system called *ASET* for ad-hoc structured exploration of text collections. Overall, we have shown that *ASET* is able to extract structured data from real-world text collections in high quality without the need to manually curate extraction pipelines. In the future, we plan to extend our system in several directions; e.g., to support more complex user queries (e.g., with joins over multiple tables) or more complex document collections.

# 17. Demonstrating *ASET*: Ad-hoc Structured Exploration of Text Collections (SIGMOD'22)


## Abstract

In this demo, we present *ASET*, a novel tool to explore the contents of unstructured data (text) by automatically transforming relevant parts into tabular form. *ASET* works in an ad-hoc manner without the need to curate extraction pipelines for the (unseen) text collection or to annotate large amounts of training data. The main idea is to use a new two-phased approach that first extracts a superset of information nuggets from the texts using existing extractors such as named entity recognizers. In a second step, it leverages embeddings and a novel matching strategy to match the extractions to a structured table definition as requested by the user. This demo features the *ASET* system with a graphical user interface that allows people without machine learning or programming expertise to explore text collections efficiently. This can be done in a self-directed and flexible manner, and *ASET* provides an intuitive impression of the result quality.


## Bibliographical Information


The content of this chapter was previously published in the peer-reviewed work "Benjamin Hättasch, Jan-Micha Bodensohn, and Carsten Binnig. 'Demonstrating ASET: Ad-hoc Structured Exploration of Text Collections'. In: *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. ACM, 2022. DOI: 10.1145/3514221.3520174". The contributions of the author of this dissertation are summarized in Section 5.2.

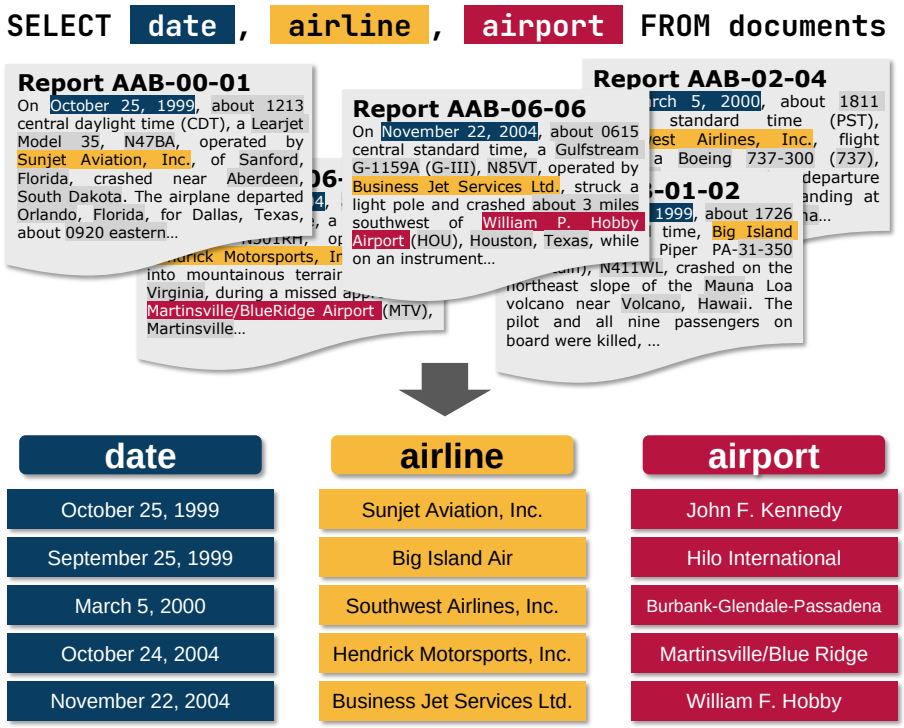



Figure 17.1.: Ad-hoc structured exploration of text collections with *ASET*: (1) a superset of information nuggets is first extracted from the texts and then (2) matched to the relevant attributes of the user query.

## 17.1. Introduction

**Motivation:** In many domains, users face the problem of having to quickly extract insights from large collections of textual documents. For example, imagine a journalist who wants to write an article about airline security that was triggered by some recent incidents of a well-known US airline. For this reason, the journalist might decide to explore a collection of textual accident reports from the *National Transportation Safety Board* in order to answer questions like "What incident types are the most frequent ones?" or "Which airlines are involved most often in incidents?" To be able to formulate such answers to their questions, they would need to extract the relevant information, create a structured data set (e.g., a spreadsheet or a database table), and analyze frequency statistics such as the number of incidents per airline.

And clearly, there are many more domains where end users want to explore textual document collections in a similar fashion. As another example, think of medical doctors who want to compare symptoms and reactions to medical treatments for different groups of patients based on the available data coming from textual patient reports. To do this, the doctor again would need to extract the relevant structured information about age, pre-existing conditions, etc. from those reports before being able to draw any conclusions.

One could now argue that extracting structured data from text is a classical problem that various communities have already tackled and several industry-scale systems already exist: DeepDive [Sa+16] or System-T [Lem+20] are examples of such systems that have developed rather versatile

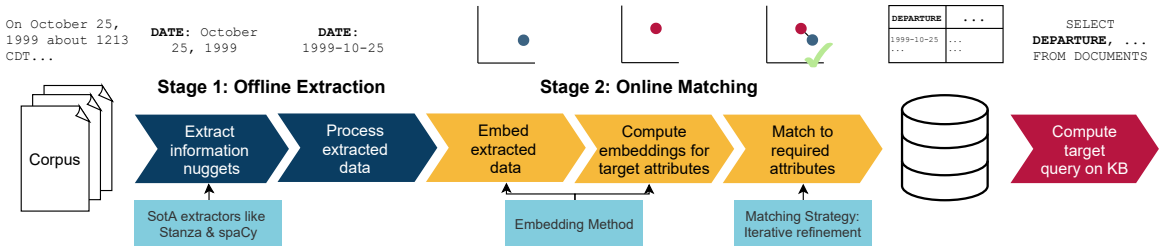



Figure 17.2.: Architecture of *ASET*: The extraction stage obtains information nuggets from the documents. The matching stage matches between the extracted information nuggets and the user's schema imposed by their query.

tool suites to extract structured facts from textual sources. However, these systems typically require a team of highly-skilled engineers that curate extraction pipelines to populate a structured database from the given text collection or train machine learning-based extraction models (that come with the additional need to create labeled training data). A major problem of these solutions is the high effort they require and, thus, it can take days or weeks to curate such extraction pipelines even if experts are involved. Even more importantly, these extraction pipelines are typically rather static and can only extract a pre-defined (i.e., fixed) set of attributes for a certain text collection. This prevents more exploratory scenarios in which users ask ad-hoc queries where it is not known upfront which information needs to be extracted or whether a new data set has to be supported on-the-fly.

**Contributions:** In this demo, we thus showcase *ASET*, a system that allows users to explore unseen text collections by deriving structured data in an ad-hoc manner; i.e., without the need to curate extraction pipelines for a particular collection. We demonstrate *ASET* in a video[1] and provide our code and data sets for download.[2]

As shown in Figure 17.1, the main idea of *ASET* is that a user specifies their information need by composing SQL-style queries or lists of attributes that define the table layout of the information requested from the text collection. For example, in Figure 17.1, the user issues an ad-hoc query to extract a table about incident dates, airlines, and airports. *ASET* then automatically populates the required table(s) with information nuggets from the documents.

*ASET* supports this ad-hoc extraction of structured information by implementing a new two-phased approach: In the *extraction phase*, a superset of information nuggets is extracted from a text collection. Afterwards, the information nuggets are matched to the required attributes in the *matching phase*. To do so, *ASET* implements a new interactive approach for matching based on neural embeddings to identify potential matches for each attribute.

Since matching arbitrarily complex document collections and user queries is a challenging task, we focus on so-called *topic-focused* document collections, in which each document provides the same type of information (e.g., an aviation incident). Thus, each document can be mapped to one row of a table. Note that this is still a challenging task since arbitrary information nuggets must be mapped to an extracted table in an ad-hoc manner.

[1] `https://link.tuda.systems/aset-video`
[2] `https://link.tuda.systems/aset`

A first version of the vision of *ASET* including an initial evaluation on two real-world data sets was published at [HBB21]. In this demo we pick up the vision of the whole application cycle presented at [Hät21]. As such, we present the integration of *ASET*'s matching procedure into a full system with an interactive user interface that is shown in our demo video (see Section 17.4). The graphical interface makes it easier to use *ASET* (especially for people without strong computer science knowledge). Moreover, compared to the original submission we also developed a new interactive matching procedure that we describe in this paper (see Section 17.3): in our new matching procedure, we leverage the human ability to quickly find patterns by presenting multiple guessed matches at once, which allows users to quickly correct wrong matches. Multiple ways to give feedback (confirm, fix, or mark that there is no match in the document) further enhance the matching's quality and flexibility.

## 17.2. Related Work

To the best of our knowledge, there is no other system working in the same way as *ASET* yet. However, there are existing approaches with similar objectives and there is previous work on some components of our approach.

**Template Filling & Information Extraction:** The goal of slot or template filling is similar to our objective [GS96], yet in contrast to our approach, these template filling approaches are specifically crafted for a fixed set of slots. Most early approaches used hand-crafted rules for that. Today, another common approach to extract a fixed set of attributes is to learn a named entity recognizer specifically for the desired entity types (e.g., [SJ19]). Named entity recognizers extract and classify a set of broad entity types like organizations, locations, or products from natural language texts. A common problem with training a named entity recognizer to extract a specific set of entity types is the substantial amount of training data required. Curating this training data manually is time-consuming and often costly, since it usually involves domain experts.

Some approaches (e.g., [Wei+19a]) attempt to avoid this problem by using active learning, which allows the learning algorithm to query the user, e.g., by selecting training instances that the user then labels by hand. Another strategy is distantly-supervised or weakly-supervised named entity recognition (e.g, [Fri+17; Lia+20]). In contrast to our system, active learning and weak supervision train named entity recognizers specifically for the desired set of entity types. Instead, we use the output of conventional named entity recognizers to populate the user-provided attributes.

**Knowledge Base Population:** Knowledge base construction or population refers to the construction or expansion of graph-structured knowledge bases (in contrast to relational tables as used in our approach). Extractive approaches like DeepDive [Sa+16], SystemT [Chi+10], and QKBFly [Ngu+17] build upon (open) information extractors like ClausIE [CG13] to derive novel knowledge from natural language texts. They also perform the adaption, cleaning, and combination stages of the knowledge base building process, which include tasks like named entity disambiguation, co-reference resolution, and canonicalization. However, most of these approaches require high manual efforts to design extraction pipelines for each knowledge base.

**Schema Matching based on Embeddings:** Schema matching refers to determining correspondences between the tables and attributes of schemata from different sources. Approaches for schema matching (e.g., [Hät+20b; Her+20]) are related to *ASET* since we frame the mapping between the information extractors' output and the user-provided list of attributes as a matching problem.

## 17.3. System Architecture

Figure 17.2 shows the architecture of *ASET*. As mentioned before, *ASET* comprises two stages: (1) the first stage extracts a superset of potentially relevant information nuggets from a collection of input documents using off-the-shelf information extractors. This step is independent of the user queries and can thus be executed offline to prepare the text collection for the ad-hoc exploration by the user. (2) At runtime, a user issues several queries against *ASET*. To answer a query, a novel interactive matching stage is executed that aims to map the information nuggets extracted in the first stage to the attributes of the user table as requested by the query.

### 17.3.1. Stage 1: Offline Extraction

As shown in Figure 17.2, the extraction stage is composed of two steps. First, it derives the information nuggets as label-mention-pairs (e.g., a date and its textual representation) from the source documents using state-of-the-art extractors. Afterwards, a preprocessing step is applied which automatically canonicalizes the extracted values, e.g., by applying normalization on temporal expressions.

**Extracting Information Nuggets:** The extractors process the collection document-by-document to generate the corresponding extractions. Clearly, a limiting factor of *ASET* is which kinds of information nuggets can be extracted in the extraction stage, since only this information can be used for the subsequent matching stage. As a default, we use the named entity recognizers from *Stanza* [Qi+20]. In general, *ASET* can be used with any extractor that produces label-mention pairs; i.e., a textual mention of an information nugget in the text (e.g., *American Airlines*) together with a natural language descriptor representing its semantic type (e.g., *Company*). Moreover, additional information about the extraction (e.g., its position in the document and the sentence around it) will also be stored and used for computing the embeddings, as we describe below.

### 17.3.2. Stage 2: Online Matching

The second stage must match the extracted information nuggets to the user table to answer the query. To do so, it first computes embeddings for the information nuggets and target attributes. Afterwards, it uses a novel interactive matching strategy that incorporates user feedback to populate the user table.

**Computing Embeddings:** A classical approach to determine a mapping between information nuggets and attributes of the user table would be to train a machine learning model in a supervised fashion. However, this would require both training time and a substantial set of labeled training data for each attribute and domain. Instead, our approach leverages embeddings to quantify the intuitive semantic closeness between information nuggets and the attributes of the user table.[3] For the attributes of the user table, only the attribute names are available to derive an embedding. To embed the information nuggets extracted in the first stage, however, we can use more information and incorporate the following signals from the extraction: (1) *label* – the entity type determined by the information extractor (e.g. *ORG*),[4] (2) *mention* – the textual representation of the entity in the text (e.g., *US Airways*), (3) *context* – the sentence in which the mention appears, (4) *position* – the position of the mention in the document.

**Matching Step:** The interactive matching step populates the user table in an attribute-by-attribute fashion. In the following, we describe the matching procedure for an attribute called *airline*:

For each information nugget, *ASET* caches a distance that represents the uncertainty with which it believes that the information nugget matches the attribute. At first, this distance is initialized as the cosine distance between the nugget's label embedding (e.g., *Organization*) and the embedding of the attribute name *airline*. Later on, this distance will be updated with the distance to the closest confirmed matching nugget,[5] allowing the system to capitalize on more signals like the textual mentions (e.g., *American Airlines*) of other matching information nuggets.

For each document with no confirmed match, *ASET* considers the information nugget with the lowest cached distance as the currently guessed match. *ASET* presents a list of the documents with the most uncertain current guesses to the user for feedback (see Figure 17.3). The user can then provide feedback for any of these guesses. They may either confirm the match, select another information nugget from the document as the match, or state that the document does not contain a matching information nugget. In case their feedback results in a confirmed match, this matching information nugget is used to update the distances on all other remaining information nuggets. Afterwards, the document is removed from the list of remaining documents, and the list of guesses presented to the user for feedback is updated.

Finally, the user may decide (based on the list of most uncertain guesses that is presented to them) to terminate the interactive matching procedure and continue with the next attribute, in which case the remaining documents' cells will be populated with their currently guessed matches.

## 17.4. Demonstration

The demonstration shows how *ASET* can be used to extract relevant information from a text collection to satisfy an information need. In the video, we use the aviation incident data set mentioned in the introduction section and demonstrate an exemplary research of a journalist

[3] We use Sentence-BERT [RG19a] and FastText [Mik+18] to compute embeddings for the natural language signals.

[4] We map the named entity recognizers' labels like *ORG* to suitable natural language expressions according to the descriptions in their specification.

[5] The distance between two information nuggets is calculated as the mean of the distances between their individual embeddings as discussed before.

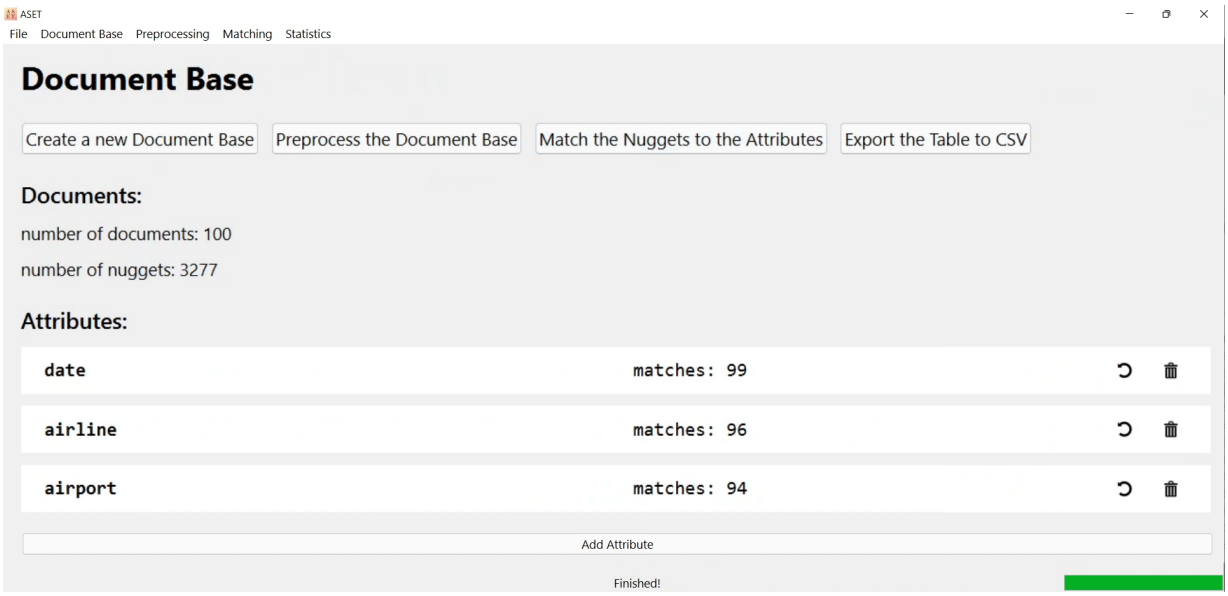




Figure 17.3.: Screenshots of the *ASET* GUI showing the matching process (where the user is asked to confirm or fix potential matches), and the summary window after successful extraction and matching.

based on this data. Other datasets we tested our system on include reports on the current COVID situation in Germany and text collections with geographic or biographical information.

The application first allows the user to load a collection of text files for processing. *ASET* then automatically extracts all named entities from the text files, applies normalization and computes the embeddings. The user is then asked to specify the attributes (column titles) of the table they need to satisfy their first information need. In the subsequent matching step, *ASET* presents a list of possible matches for each attribute. The user can confirm some of them, correct wrong matches by choosing the relevant extraction or mark if the required attribute does not occur in the given file (see Figure 17.3). *ASET* continuously updates the list of guessed matches during this interactive feedback phase, which allows the user to quickly identify (wrong) entries that stand out and get an impression of the quality already achieved. Once the user is satisfied with the quality of the matches, they continue with the next attribute.

Afterwards, the resulting table can be exported to a spreadsheet, as a SQLite table, or a Pandas Dataframe for further investigation. Alternatively, the user can change the target scheme by adding or removing attributes. *ASET* will leverage existing matching results to ensure that the interactive matching does not have to be repeated for an attribute (unless the user explicitly enforces it). All intermediate results are stored, allowing the user to pick up their analysis again on another day or to pass it to another person for further investigation.

In our demo video[6] we show how *ASET* can be used to first create a table of date, airline, and airport for each accident from a collection of aviation incident reports with only few iterations of feedback on possible matches. This table can then be opened in a spreadsheet software to filter the accidents down to those from the last years or from certain airlines. Afterwards, *ASET* is used to amend the registration numbers of the aircraft to the table, which can be used to further investigate some of the cases.

## 17.5. Conclusion & Future Work

In this demo, we showcased a new system called *ASET* that allows users to extract structured data from text collections in an ad-hoc manner without the need to manually design extraction pipelines. We presented a graphical interface that allows people without any machine learning or programming experience to use the system and described this usage. In the future, we plan to extend *ASET* in several directions; e.g., to support more complex user queries containing JOINs or more heterogeneous document collections.

[6] `https://link.tuda.systems/aset-video`

# 18. WannaDB: Ad-hoc SQL Queries over Text Collections (BTW'23)

> Just tell it what you want, what you really, really want


## Abstract

In this paper, we propose a new system called *WannaDB* that allows users to interactively perform structured explorations of text collections in an ad-hoc manner. Extracting structured data from text is a classical problem where a plenitude of approaches and even industry-scale systems already exist. However, these approaches lack in the ability to support the ad-hoc exploration of texts using structured queries. The main idea of *WannaDB* is to include user interaction to support ad-hoc SQL queries over text collections using a new two-phased approach. First, a superset of information nuggets from the texts is extracted using existing extractors such as named entity recognizers. Then, the extractions are interactively matched to a structured table definition as requested by the user based on embeddings. In our evaluation, we show that *WannaDB* is thus able to extract structured data from a broad range of (real-world) text collections in high quality without the need to design extraction pipelines upfront.


## Bibliographical Information


The content of this chapter was previously published in the peer-reviewed work "Benjamin Hättasch, Jan-Micha Bodensohn, Liane Vogel, Matthias Urban, and Carsten Binnig. 'WannaDB: Ad-hoc SQL Queries over Text Collections'. In: *Datenbanksysteme für Business, Technologie und Web (BTW 2023), 20. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 06.-10, März 2023, Dresden, Germany, Proceedings*. Volume P-331. LNI. Gesellschaft für Informatik e.V., 2023. DOI: 10.18420/BTW2023-08". The contributions of the author of this dissertation are summarized in Section 5.2.

```
SELECT report_date WHERE incidence_rate > 500;
SELECT region, AVG(incidence_rate) GROUP BY region HAVING AVG(incidence_rate) > 500;
SELECT AVG(vaccinated_twice) WHERE report_date > 21-01-01 AND report_date < 21-02-01;
```

Figure 18.1.: Exemplary ad-hoc information needs phrased as SQL-like queries in *WannaDB*. Two classes of ad-hoc queries are supported: Queries that extract facts from individual documents (e.g., first query) as well as queries that involve aggregation and grouping (e.g., the latter two queries).

## 18.1. Introduction

A question like "What were the days with a COVID-19 incidence rate higher than 750 in Germany?" can be answered with a simple SQL query if the relevant information is present in a database. Yet, in case there are only written (i.e., textual) reports available such as those published by governmental organizations like the RKI in Germany,[1] the situation is much more complex: answering such queries over collections of textual documents that each contain only a part of the information needed requires that first the relevant attributes are extracted from each document, before they are stored in a structured form (i.e., a spreadsheet or a database table) in order to make them available for structured queries.

One could now argue that extracting structured data from text is a classical problem for which there is a plethora of approaches and where even several industry-scale systems already exist: for example, DeepDive [Sa+16] that was acquired by Apple or System-T [Lem+20] from IBM are such systems that have developed rather versatile tool suites to extract structured facts from textual sources. However, these systems require a team of highly-skilled engineers that compile extraction pipelines, which often includes training particular machine learning models, and then populate a structured database from the given text collection. And even more importantly, the resulting extraction pipelines are typically static and can only be used to extract a pre-defined (i.e., fixed) set of attributes and tables for a certain text collection. This prevents exploratory scenarios where users can ask ad-hoc queries regardless of whether a pipeline has been set up to extract the attribute or not.

Hence, being able to ad-hoc execute SQL-like queries over a text collection without the need to manually compose extraction pipelines would be a major step forward compared to existing approaches for structured data extraction from text. Use cases with needs for such ad-hoc structured querying of unstructured text can be found in various domains beyond the example mentioned before, e.g., data scientists together with medical doctors looking for new insights through medical reports or data journalists examining hundreds of documents as part of their investigations. Structured queries provide a higher expressiveness (e.g., aggregation and filtering operations), and more rigorousness in the calculation of the results compared to the usage of natural language queries in classical question answering systems.

**Contributions.** In this paper, we hence propose *WannaDB*, a system that can execute SQL-like queries on text collections in an ad-hoc manner. Examples for queries that *WannaDB* supports can be found in Figure 18.1. Overall, *WannaDB* supports two classes of queries: *(1) Ad-hoc Fact*

[1] `https://www.rki.de/DE/Content/InfAZ/N/Neuartiges_Coronavirus/Situationsberichte/Gesamt.html`

*Queries:* queries that extract facts from text documents to construct table rows. This also involves applying filter predicates and projection operations, as shown by the first query in Figure 18.1. *(2) Ad-hoc Aggregate Queries:* queries that in addition involve aggregations and grouping over multiple documents as shown by the two other queries in Figure 18.1, which come with additional challenges like named entity disambiguation/cross-document co-reference resolution that we discuss later in this paper. *WannaDB* can therefore directly produce tables stating information that is not explicitly mentioned in the documents and hence not discoverable by pure extraction or search approaches. To enable such ad-hoc SQL queries over a given text collection, *WannaDB* implements a novel extraction and querying pipeline that builds on two key ideas:

The first key idea of *WannaDB* is that, different from existing approaches which aim to extract information for a specific (i.e., fixed) information need from a given text collection, *WannaDB* instead implements a *holistic extraction* approach that aims to extract a wide spectrum of information from a given text collection (called information nuggets in the sequel). For this holistic extraction, *WannaDB* implements a framework approach and relies on a set of different general-purpose extraction methods, such as approaches for named-entity recognition. Moreover, during extraction, *WannaDB* computes embeddings for all the information nuggets, taking several signals such as the textual mentions itself, and the position in the text into account.

As a second key idea, to answer ad-hoc queries on top of the extracted information nuggets, *WannaDB* implements a novel *interactive matching* approach that aims to map the information nuggets to the information needs specified by the user in form of an SQL query: embeddings of the extracted information nuggets together with the embeddings of the query attributes are used to decide which information nuggets qualify for answering the query. For this matching, *WannaDB* requests feedback from the user whether certain information nuggets are the correct values for the required query attributes. The system carefully selects these requests to minimize the amount of required feedback. The query attributes can be of a much finer granularity than the labels of the extraction approaches used in the first stage (e.g., `airline` instead of `ORG`) and *WannaDB* can even distinguish between similar attributes with just a small semantic difference (e.g., the amounts of people vaccinated once and twice).

While other approaches that can extract tables from text such as learned sequence-to-sequence models [WZL22] often suffer from a phenomenon called hallucination (i.e., they generate values that are not in the actual source document), our approach can guarantee that the contents of the produced result tables always originate from the queried documents. Moreover, compared to learned question answering approaches, *WannaDB* can perform numerical reasoning on the data without the need to rely on the limited mathematical abilities [Hen+21] of a language model.

In order to evaluate the abilities of *WannaDB*, we conduct a wide range of experiments on text collections from different domains ranging from aviation reports over daily COVID-19 situation reports to multiple text collections created from Wikipedia that cover different categories (Nobel laureates, countries, and skyscrapers). We show that *WannaDB* not only outperforms other baselines that can be used for ad-hoc query answering on text collections, but is also competitive with approaches that are trained or refined on domain-specific data. Moreover, our evaluation shows that typically only a few interactions per query attribute are sufficient to answer a query over hundreds or thousands of source documents. Overall, answering an SQL query over text documents with *WannaDB* (by providing minimal interactive feedback) only takes a few minutes, compared to hours and hours of manually extracting information or refining an extraction pipeline

without *WannaDB*. Finally, to make the results reproducible, we will make our source code and the data sets used for evaluation available at `https://link.tuda.systems/wannadb`.

**Outline.** Next, we describe the functions of *WannaDB* in an exemplary usage scenario, before we explain the different components in Section 18.3. In Section 18.4, the algorithms behind the interactive components are discussed in further depth, followed by a short overview of the current limitations in Section 18.5. We provide an evaluation of *WannaDB* in Section 18.6 and an overview of existing and related work in Section 18.7, before we conclude in Section 18.8.

## 18.2. Exemplary Usage

In this exemplary usage scenario, we aim to show how *WannaDB* can be used to satisfy an information need based on a text collection. Imagine, e.g., a data journalist who just obtained a large collection of airline incident reports and is now looking for noticeable events, like a high rate of incidents for a certain carrier or airport. They use *WannaDB* for that purpose. The data journalist starts by loading the collection of text files into *WannaDB* for processing and triggers the pre-processing of the files, a process that needs to be done only a single time for each text collection.

Next, the data journalist enters an SQL-like query as a starting point for their exploration (e.g., `SELECT airline, airport, COUNT(*) GROUP BY airline, airport`). As there is no pre-existing table yet, the FROM-part of a typical SQL query can be omitted, simplifying the query syntax. After entering the query, *WannaDB* presents a list of possible matches for each required attribute (e.g., airline) found in texts of the collection, as shown in Figure 18.2. Not all the found matches will be correct right away, therefore *WannaDB* relies on some user input to adjust the results. The data journalist confirms a few of the correctly found matches, corrects wrong matches by choosing the relevant extraction or marks if the required attribute does not occur in a given text (see Figure 18.2). Meanwhile, *WannaDB* continuously updates the list of all guessed matches during this interactive phase, leveraging the feedback. The user interface allows to quickly identify entries that stand out and get an impression of the quality already achieved. Once the data journalist is satisfied with the quality of the matches, they continue with the next attribute of their query.

After all attributes are processed, *WannaDB* will execute the query on the resulting table. If the query contains grouping operations, the data journalist might be asked again for some interactive feedback (e.g., to confirm that *Lufthansa* and *LH* refer to the same airline, but *LHS* does not). *WannaDB* will again try to transfer this feedback to other rows. In the end, the data journalist will receive an answer to their query and can export the resulting table to a spreadsheet, an SQLite table, or a Pandas Dataframe for further investigation. If they have further queries to submit to *WannaDB*, the interactive matching process only needs to be repeated for new attributes, as *WannaDB* leverages existing results from previous queries.

## 18.3. System Overview & Architecture

In this section, we describe the architecture of *WannaDB*. It consists of two stages: an offline stage to extract information nuggets (i.e. short information-bearing text snippets), followed by

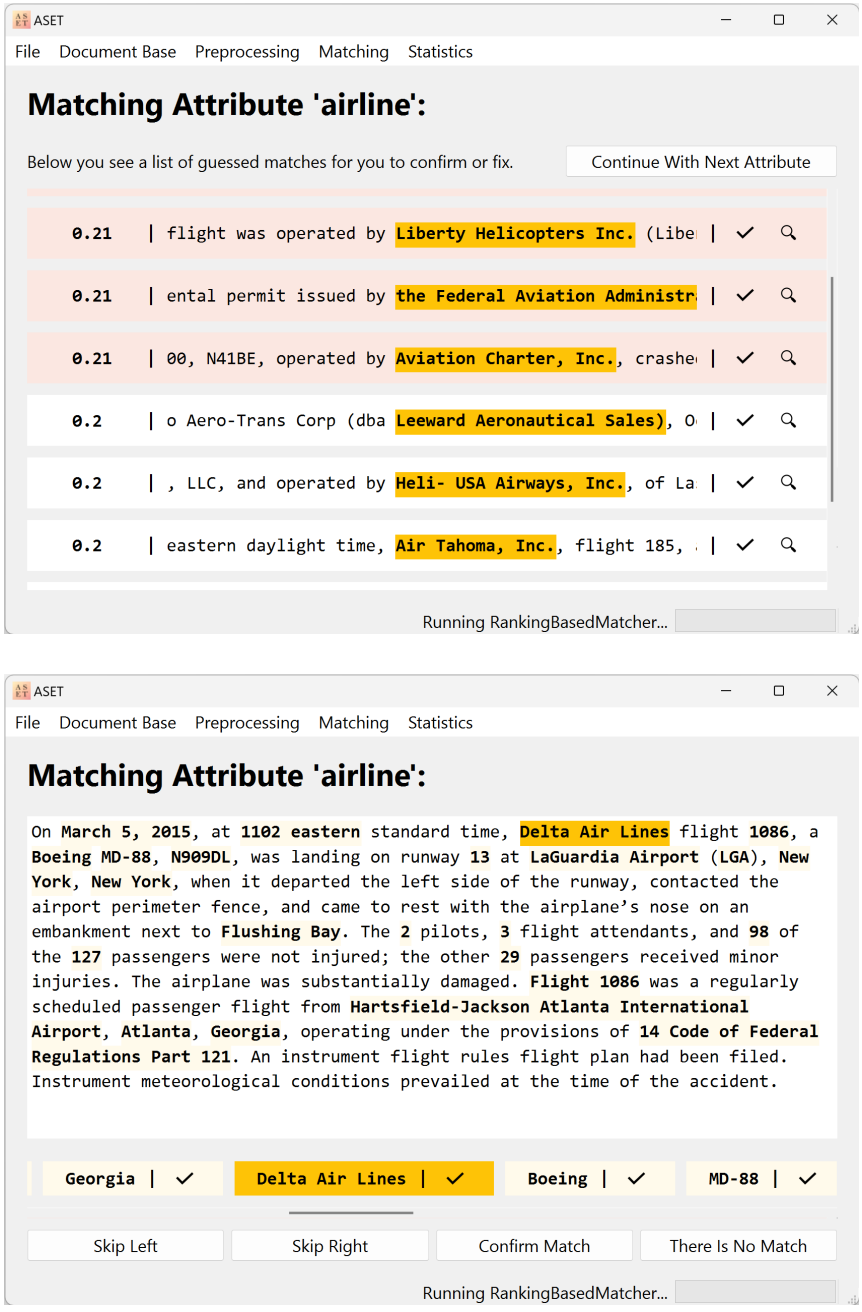



Figure 18.2.: Graphical user interface of *WannaDB*, more details can be found in our SIGMOD'22 demo [HBB22]. Left: potential matches over and under the threshold are shown, the user is asked to either confirm or fix them. Right: Inspect a document and fix by selecting the correct match.

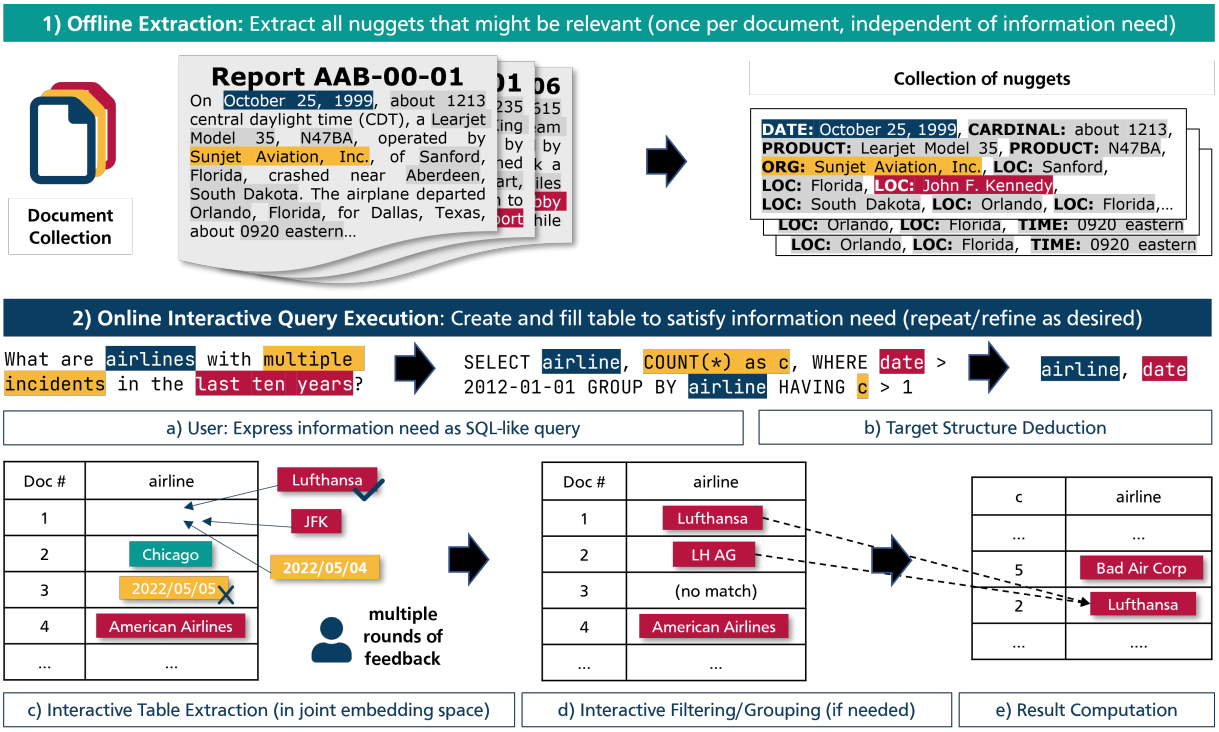



Figure 18.3.: Architecture & exemplary usage: The offline extraction phase obtains information nuggets from the documents. The online phase then infers the required structure from a query, matches between the extracted information nuggets and the user's schema, performs the grouping and executes the query.

the interactive stage to answer the query by table extraction and if required interactive filtering or grouping. The overall workflow is visualized in Figure 18.3. Here, we give an overview of both stages and the relevant components of *WannaDB*. More details of the table extraction as well as grouping and filtering, which are the main contributions of *WannaDB*, are described in Section 18.4.

### 18.3.1. Stage 1: Offline Extraction

In the first stage we employ off-the-shelf information extractors to extract a superset of potentially relevant information nuggets (e.g., named entities) from the given text collection. This step is independent of user queries and can thus be executed offline to prepare the text collection for ad-hoc exploration by the user. The extractors process the collection document-by-document to generate the corresponding extractions. Clearly, a limiting factor of *WannaDB* is which kinds of information nuggets can be extracted in the extraction stage, since only this information can be used for the subsequent matching stage. As a default, we use named entity recognizers from *Stanza* [Qi+20] and *spaCy* [Hon+20]. In general, *WannaDB* can be used with any extractor that produces label-mention pairs; i.e. a textual mention of an information nugget in the text (e.g., *American Airlines*) together with a natural language descriptor representing its semantic type called label (e.g., *Company*). Moreover, additional information about the extraction (e.g., its position in the

document and the surrounding sentence) is also stored and used for computing the embeddings, as we describe below.

After extraction, the information nuggets are pre-processed to derive their actual data values (i.e., a canonical representation, e.g., for timestamps) from their mentions. For this we also rely on state-of-the-art systems for normalization [Man+14]. The nuggets are then represented based on the following signals: (1) *label* – the entity type determined by the information extractor (e.g. *Company*),[2] (2) *mention* – the textual representation of the entity in the text (e.g., *Lufthansa*), (3) *context* – the sentence in which the mention appears, (4) *position* – the position of the mention in the document. Each information nugget representation comprises embeddings for the individual signals (1-4). We compute semantic representations for the natural language signals using FastText [Mik+18] (1), Sentence-BERT [RG19a] (2) and BERT [Dev+19] (3) and normalize the position by dividing it by the document length.

### 18.3.2. Stage 2: Interactive Query Execution

At runtime, a user issues queries and interacts with the system. *WannaDB* infers the table structure required to answer a query, and employs a novel interactive matching stage to map the information nuggets extracted in the first stage to the required query attributes.

**Interactive Table Extraction.** The first step of the interactive query execution of *WannaDB* is the interactive table extraction from the text documents. In this step, a table with attributes is filled by *WannaDB* to answer a given user query. The required table structure is automatically inferred from the user's SQL query. *WannaDB* checks which attributes are mentioned explicitly as attributes to return, and as part of aggregation operations, or implicitly in filter predicates or group-by statements. Then, *WannaDB* starts to fill the table with the derived schema by executing the interactive table extraction algorithm.

In the interactive table extraction, the user interacts with *WannaDB* in order to fill the required attributes of the result table with the information nuggets extracted before. To find matching nuggets, *WannaDB* first computes embeddings for the target attributes similar to the ones computed for the information nuggets in the offline phase.

A classical approach to determine a mapping between information nuggets and attributes of the user table would be to train a machine learning model in a supervised fashion to classify to which attribute the extracted information nugget should be mapped to. However, learning such a classification model would require a substantial set of labeled training data for each attribute and thus prevent ad-hoc queries. Instead, our approach leverages embeddings to quantify the intuitive semantic closeness between information nuggets and the attributes of the user table. For the attributes of the target table, only the attribute names are available to derive an embedding, while for the extracted nuggets we can make use of more information as we described above.

*WannaDB* therefore employs a novel interactive matching strategy that incorporates user feedback and operates in the joint embedding space of nuggets and target attributes. This strategy works in an attribute-by-attribute fashion and collects user feedback (e.g., confirming or correcting a possible match). *WannaDB* uses distances between possible and confirmed matches to populate the

[2] We map the named entity recognizers' labels like *ORG* to suitable natural language expressions according to the descriptions in their specification.

remaining cells. This process is steered by carefully selecting potential matches that are presented to the user for feedback to reach a high matching quality with as little feedback as possible.

**Interactive Filtering & Grouping.** After the interactive table extraction step, *WannaDB* executes the interactive filtering and grouping stage for answering a user query. Remember, *WannaDB* has the aim to work on text collections from domains without pre-existing resources like refined language models or custom knowledge bases. Grouping and filtering the extracted table thus is challenging, since it is filled with mentions from the text directly, hence applying these operations might lead to faulty query results if entities are not correctly resolved: e.g., the table might contain entries such as *Deutsche Lufthansa* and *German Lufthansa Airline* which both refer to the same entity. Applying `GROUP BY` or a `WHERE` directly on such an extracted table would return multiple lines (i.e., one for each different mention even though they refer to the same entity). *WannaDB* therefore again uses interaction to perform those operations on the level of embeddings instead of string representations, as will be described in detail in the next section.

## 18.4. Interactive Query Execution

*WannaDB* introduces novel embedding-based algorithms for interactive table extraction as well as filtering and grouping. In this section, we describe these algorithms in further detail (see Figure 18.4 for a pseudocode representation).

### 18.4.1. Interactive Table Extraction

In the interactive table extraction stage, *WannaDB* populates the attributes of the table one by one. To fill the cells of a certain attribute, *WannaDB* aims to select one matching information nugget from each of the documents. To do so, *WannaDB* associates each information nugget with a *cached distance* that corresponds to the certainty with which it believes that the nugget matches the attribute. For each document, *WannaDB* considers the information nugget with the lowest cached distance as the document's currently *guessed match*. Furthermore, *WannaDB* uses a distance *threshold* for each attribute to decide when a cell should be left empty instead. The details of how this threshold is calculated and interactively adapted are explained in Section 18.4.2. The overall procedure of the table extraction is shown in Figure 18.4.

In the beginning, each nugget's cached distance is initialized as the cosine distance between the nugget's label embedding (e.g., *Organization*) and the embedding of the attribute name (e.g., *Airline*) (Figure 18.4, line 2-3). After initialization, the interactive feedback phase starts. *WannaDB* presents a ranked list of documents with their currently guessed matches to the user for feedback (see Figure 18.2) and will continuously update the list after every given feedback. This allows the user to quickly identify (incorrect) entries that stand out and to get an impression of the quality already achieved. The ranked list is centered around the threshold and thus hopefully shows both correct guesses with a low certainty, and incorrect guesses, where *WannaDB* would profit most from feedback.

The user can then provide feedback for any of these guesses (line 7): they may either confirm the guess, select another information nugget from the document, or state that the document does not contain a matching information nugget. In case their feedback results in a confirmed

match, this matching information nugget is used to update the cached distances of all other remaining information nuggets (line 13-16). To compute the distance between two information nuggets, *WannaDB* calculates the mean of the cosine distances between their individual signal embeddings. The distance updates ensure that a nugget's cached distance is always the distance to the closest confirmed match. Considering distances between information nuggets allows *WannaDB* to capitalize on more signals like the textual mentions (e.g., *American Airlines*) of other matching information nuggets.

Next, *WannaDB* updates the documents' currently guessed matches by selecting the information nuggets with the lowest cached distances (line 21). Finally, *WannaDB* then adjusts the threshold accordingly (see Section 18.4.2 for more details). Moreover, the user can at any time decide to terminate the interactive feedback phase and continue with the next attribute. All remaining documents' cells without explicitly confirmed matches will then be populated with their currently guessed matches (line 24-28) if there is at least one with a distance that is low enough (i.e., below the threshold).

### 18.4.2. Threshold Adjustment

*WannaDB* uses a threshold for two purposes: (a) to decide when it is better to leave a cell empty than to use a very unlikely guess (mostly because the desired value is not mentioned in the document) and (b) to select guesses to present to the user where feedback will have as much effect as possible. This threshold is automatically tuned during the runtime of *WannaDB* to fit the data at hand. Given the approximate query setting *WannaDB* is built for, we decided to use a common threshold for all regions forming in the embedding space instead of individually tuning it, to keep the number of interaction cycles low.

The adjustment of the threshold is shown in Figure 18.4 (line 30-47). The general idea is to incorporate the additional knowledge gained from the user confirming a nugget even though it was above the threshold or correcting an entry below the threshold. This feedback action will only affect a certain nugget directly, but other similarly well fitting nuggets from other documents might still be accepted or discarded wrongly because of the threshold, which is therefore carefully adapted after feedback actions: If the user confirms a nugget from the ranked list that is above the threshold, all nuggets between the threshold and this nugget should be considered as a good guess. In the case that any of the nuggets is still above the threshold after the calculation of the new distances, the threshold is adapted accordingly. In contrast, if the user states that for a nugget with a distance below the threshold there is no match in the document, the threshold is decreased to also exclude other matches that are in the list above the nugget if necessary. The threshold is only adapted in these two cases, where implicit hints about the quality assessment by the user can be incorporated.

### 18.4.3. Interactive Filtering & Grouping

In the following, we explain how interactive grouping is supported in *WannaDB* to tackle the problem of different surface forms for the same entries. Filtering works similarly, but we omit the details due to space limitations.

```
for attribute in query.attributes: # Process each attribute separately
  for nugget in all_nuggets:
    nugget.distance = compute_distance(attribute, nugget) # Compute initial
    ↪ distances

  while interactive_feedback_phase:  # Interactively get user feedback
    ranked_list = make_ranked_list(threshold, documents)
    feedback = get_user_feedback(ranked_list)
    match feedback:
      # Positive feedback (confirmation or manually correction):
      case ConfirmNugget(document, confirmed_nugget):
        # Mark this particular cell as manual confirmed...
        set_match(document, confirmed_nugget)
        # ... and update distances for all nuggets based on user feedback
        for nugget in all_nuggets:
          new_distance = compute_distance(nugget, confirmed_nugget)
          nugget.distance = min(new_distance, nugget.distance)
      # Negative feedback:
      case NoMatchInDocument(document):
        # Direct effect only on the given document...
        leave_empty(document)
    update_guessed_matches(documents)
    adjust_threshold(feedback)  # ... but both feedback types can have effects
    ↪ indirectly through threshold adjustment on other document's rows, too

  for document in documents:  # Only consider values up to a given maximum distance
    if current_guess(document).distance < threshold:
      set_match(document, current_guess(document)) # compute final result table
    else:
      leave_empty(document)

def adjust_threshold(feedback):  # Feedback can be further exploited in certain
↪ cases
  match feedback:
    case ConfirmNugget(document, confirmed_nugget):
      if confirmed_nugget.distance > threshold:
        increase_threshold(confirmed_nugget)
    case NoMatchInDocument(document):
      if current_guess(document).distance < threshold:
        decrease_threshold(document)

def decrease_threshold(document):  # Consider fewer matches as valid (especially
↪ those above last marking as incorrect that are currently accepted nevertheless)
  nuggets = ranked_list.between(threshold, document)
  min_dist = min(n.distance for n in nuggets)
  threshold = min(min_dist, threshold)

def increase_threshold(confirmed_nugget):  # Consider more matches as valid
↪ (especially those below last confirmation that are currently discarded because
↪ of the threshold)
  nuggets = ranked_list.between(confirmed_nugget, threshold)
  max_dist = max(n.distance for n in nuggets)
  threshold = max(max_dist, threshold)
```

Figure 18.4.: Pseudo-Code representation of our interactive algorithm for table extraction, including threshold adjustment.

To resolve entities correctly, the interactive grouping algorithm is based on agglomerative clustering using the distances between the information nugget embeddings for an attribute. Entries with the same string representation are merged without interaction. For the remaining ones, the different signals from the extraction phase are utilized. *WannaDB* presents all distinct members of two clusters that should potentially be merged to the user and asks them to confirm whether these all describe the same entity. If that is the case, the clusters are merged and the distances are recalculated. To minimize the amount of necessary interactions with the user, *WannaDB* does not always ask for the pair of clusters with the lowest distance, but chooses a pair with a higher distance, using a step size that is adapted based on the last interactions. If the user confirms the equivalence of the candidates, not only that pair but also those with a substantially lower distances are merged. If the entries of the merging candidates are marked as different, *WannaDB* continues to search for a better threshold for the distance between clusters using a binary search pattern.

## 18.5. Current Limitations of *WannaDB*

In order to build a system that can quickly compute query results on various domains, we introduce two limitations: First, *WannaDB* currently can only answer single-table queries on top of document collections; i.e., we extract one table per document collection where each row of the table corresponds to one document. However, this is not a severe limitation, since the extracted table can be seen as the materialized result of a join. *WannaDB* will extract a wide table (e.g., containing information about an incident itself but also the airlines and airports involved)—but only with the attributes that are required for a given query.

Second, the results produced by *WannaDB* are always approximate. While *WannaDB* can achieve a high F1-score for all attributes (as we will show below), query results might be incomplete (i.e., values of attributes might be missing) or the extracted values might be dirty (e.g., a group-by statement might result in two instead of one group due to a not fully correct clustering). However, we believe that the query results of *WannaDB* are still of high value to users, providing them with a trend and allowing them to decide if something interesting is contained in the document collection in a short time.

## 18.6. Experimental Evaluation

In this evaluation, we aim to show the abilities of *WannaDB* on text collections from different domains. We will demonstrate the end-to-end performance, compare our table filling approach to non-interactive and learned models, and evaluate the effects of interaction, and the scalability of *WannaDB*. To the best of our knowledge, there is no system working like *WannaDB* yet. Therefore, we cannot compare our results end-to-end with existing systems. As the whole task of running SQL queries over text collections is quite complex, there is no simple baseline for comparison either. However, we evaluate the components of our approach individually, and show that *WannaDB* performs better compared to various baselines. We perform our evaluation on three data sets from very different domains. Each of them consists of a document collection as well as a ground-truth extraction of structured data that we can use to evaluate the results of executing ad-hoc queries with *WannaDB*.

**Aviation.** The first data set is based on aviation accident reports published by the United States National Transportation Safety Board (NTSB).[3] Each report documents a severe aviation accident and provides details like the prevailing circumstances, probable causes, conclusions, and recommendations. For the experiments, we use the executive summaries that the NTSB publishes with each report. As a ground-truth, we compiled a list of twelve attributes based on frequently occurring facts from the summaries. We then manually created annotations that capture where the summaries mention the attributes' values. The final data set comprises $100$ annotated documents and a table which provides the ground-truth structured data for all attributes.

**COVID19.** The second data set is based on the German RKI's daily reports outlining the situation of the Covid-19 pandemic in Germany.[4] We again used the summaries of the full documents, which contain information like the number of new laboratory-confirmed Covid-19 cases or the number of Covid-19 patients in intensive care. We compiled a list of seven all-numeric attributes, which is in particular challenging compared to string-valued attributes, since these are harder to separate into different attributes in the embedding space. As a ground-truth for the experiments, we manually annotated the occurrences of all these seven attributes again in $100$ reports.

**T-REx: Countries, Nobel & Skyscrapers.** In addition to the data sets before that we explicitly created for evaluating *WannaDB*, we adapted the T-REx data set [ElS+18] that was also used in other papers. The original data set consists of 11 million Wikidata triples aligned with $3.09$ million Wikipedia abstracts. We extracted three subsets based on article categories from different domains: *Countries* consists of 187 documents with three annotated attributes, *Nobel* challenges to extract four attributes (date of birth and death, field of work and country) for 209 Nobel Prize laureates, and *Skyscrapers* is by far the largest data set with 2683 documents containing annotations for three attributes. All these data sets are quite sparse, since most of the time only a subset of the attributes is contained in a document. Therefore, this data set is valuable to test how well *WannaDB* can work when information in documents is missing.

**Metrics.** As a main metric, we report the F1 score in most experiments (values between 0 and 1, higher is better) as an aggregated value that incorporates both the precision (i.e., the correctness of the table cell values) of our approach and its recall (i.e., the extent to which table cells are filled as expected). The F1 scores we report are calculated based on the ground truth and predictions in the filled tables. We thereby consider cells (i.e., an attribute value) as true positives when they are correctly filled with information from the text corresponding to that row, and as true negatives when they are correctly left empty, in case the required information is not present in the corresponding text. False positive predictions occur, when a cell is filled incorrectly. False negatives occur when a cell is left empty that should have been filled with data from the text, and also for incorrectly filled cells, as the correct nugget has not been found.

### 18.6.1. Exp. 1 – End-to-end Queries

To provide an indication of how *WannaDB* works end-to-end, we perform a qualitative analysis on queries involving aggregation and grouping over multiple documents before we later-on show quantitative results for *WannaDB*. For the experiments, we assume that a user always provides

[3] `https://www.ntsb.gov/investigations/AccidentReports/Pages/Reports.aspx?mode=Aviation`

[4] `https://www.rki.de/DE/Content/InfAZ/N/Neuartiges_Coronavirus/Situationsberichte/Gesamt.html`

T-REx nobel laureate:

SELECT country, COUNT( * ) AS count
GROUP BY country
SORT BY count

P: 1.0000
R: 0.5714
MJI: 0.4775

| country | count *(correct)* |
|---|---|
| American | **88** *(84)* |
| Soviet | **9** *(10)* |
| Swiss | **5** *(4)* |
| Japan | **3** |
| Germany | **2** *(3)* |
| ... | ... |

T-REx country:

SELECT continent, COUNT( * ) AS count
GROUP BY language
SORT BY count

P: 0.7500
R: 1.0000
MJI: 0.5004

| continent | count *(correct)* |
|---|---|
| Africa | **33** *(34)* |
| Europe | **19** *(20)* |
| Asia | **12** *(14)* |
| South America | **11** *(13)* |
| North America | **4** *(5)* |
| ... | ... |

Figure 18.5.: End-to-end results for two queries executed on T-REx data sets. The tables show the first five rows of the resulting table (one attribute column filled by *WannaDB* plus aggregation results). The bracketed values indicate the ground truth values. Additionally, precision (P) and recall (R) computed at cluster level, and mean Jaccard Index (MJI) averaged over all clusters are reported.

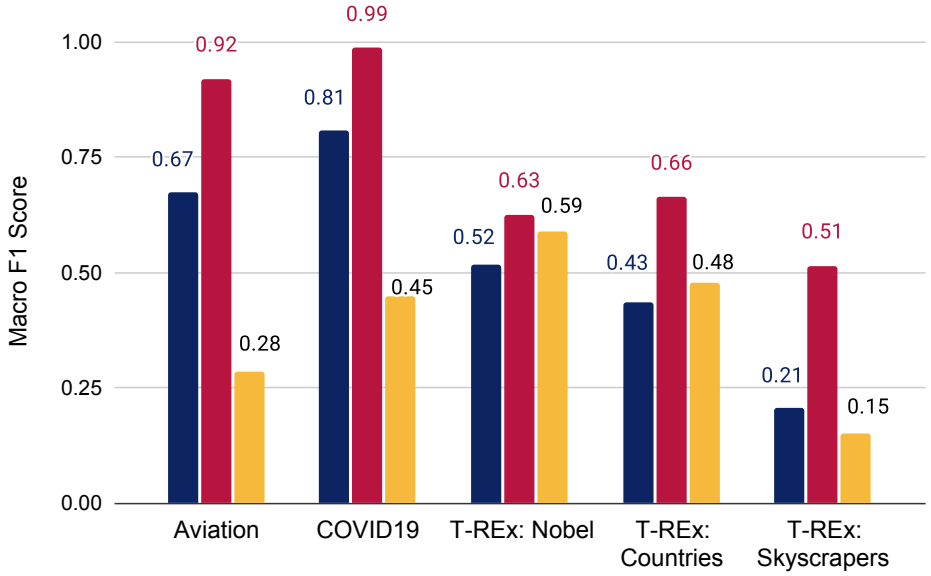
correct feedback for *WannaDB* to execute the matching of extractions to query attributes. However, we do not expect optimal feedback, i.e., the simulated feedback actions are not chosen in a way to maximize speed of convergence. We report the results after using 20 simulated user interactions (i.e., 20 times confirming an extraction or choosing an alternative one as a match for a query attribute). We discuss the interaction effort that is needed for *WannaDB* to perform extractions in a separate experiment.

Figure 18.5 shows the first five rows of the query results for two aggregation queries executed on the *T-REx Nobel* and the *T-REx Countries* data sets. Additionally, precision and recall, as well as a numeric score of the correctness of the clusters, can be seen. While *WannaDB* delivered the correct values for the group-by operation, the aggregation (COUNT) deviates slightly from the ground-truth. The reason is that for some documents, *WannaDB* could not extract the requested information. As such, the results of *WannaDB* can be seen as an approximation of the true query result that can be used for quickly gaining (initial) insights into text collections. Moreover, it is important to note that existing extraction baselines—that in contrast to *WannaDB* do not support ad-hoc queries—also do not provide perfect extractions (as we show in the following experiments).

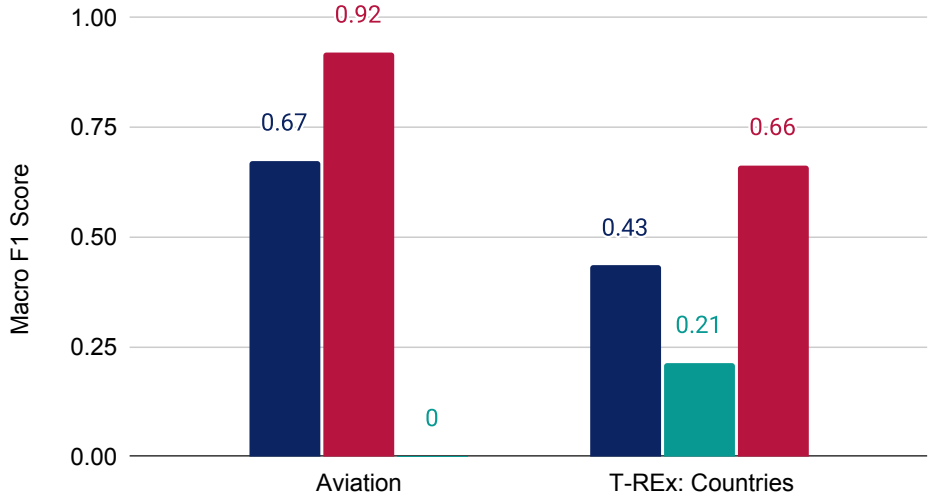
### 18.6.2. Exp. 2 – Interactive Table Extraction

In the second experiment, we quantitatively evaluate how well *WannaDB* can fill a table specified by a user's query with information from the texts. For this, we focus on the quality of the interactive table extraction, which is the most important step for *WannaDB* to provide high-quality query results; i.e., if the table extraction is not able to provide high accuracy, grouping and filtering will also not be able to provide high accuracy. For showing the quality of *WannaDB*, we run the experiments in this section on all three data sets (Aviation, COVID19 and T-REx).

**Baselines.** To put the results of *WannaDB* into perspective, we compare it to two baselines based on BART [Lew+20]. BART is a state-of-the-art pre-trained transformer model, with a high capacity to learn text-based tasks with minimal overhead of fine-tuning. Its robust architecture outperformed older transformers, especially on tasks like question answering. We use the openly available *bart-large* model from the Huggingface [Wol+19] library and formulate information extraction




(a) Table filling results. It can be seen that ■ *WannaDB* performs comparable to the ■ BART model trained explicitly on the data (fine-tuned per topic individually) and outperforms the generic ■ BART model (fine-tuned on SQuAD) most of the time.




(b) Upper Baseline BART models show low generalization abilities on unseen data sets—the performance of a ■ model trained on one data set drops drastically when applied to ■ the other data set. ■ *WannaDB* for comparison.

Figure 18.6.: Text-to-Table Results

for individual query attributes as a sequence-to-sequence task (i.e., the input is a text document and the output is the structured data extracted from the text). For fine-tuning BART for the information extraction task on a particular data set (i.e., transforming a text into a table) we use the following procedure: We split each data set into 75% that we use as train set for fine-tuning, 15% as validation set and 10% as a holdout test set. We then fine-tune one BART model on each data set for 50 epochs with a learning rate of $1e-5$ and batch size of 2, which yielded the best performance in our experiments. Moreover, we select the best checkpoint from the 50 epochs based on the validation set for evaluation. Important to note here is that the resulting fine-tuned BART models are an upper baseline for *WannaDB*, as they are trained supervised on the annotated data and all possible query attributes; i.e., with this baseline we do not test the ad-hoc scenario that we envision for *WannaDB*, but instead assume that all query attributes are known in advance.

For comparing *WannaDB* to a baseline that supports ad-hoc queries on a new (unseen) text collection, we use a second variant that is also based on BART but not pre-trained on the particular data set and query attributes. For this baseline, we instead use a BART model[5] that is already fine-tuned for extracting structured information from the SQuAD 2.0 data set [RJL18].[6] For the experiment, we use this fine-tuned model on an unseen data set and extract attributes that the model has not seen during fine-tuning.

**WannaDB vs. Baselines.** The results of *WannaDB* in comparison with the two BART models are shown in Figure 18.6a. For *WannaDB*, we report the median over 20 randomized runs, and again use 20 simulated user interactions per attribute. As baselines, we use the two variants of BART discussed before.[7] BART models fine-tuned per data set (red bars) are able to achieve high F1 scores on the data and query attributes they were trained on, outperforming *WannaDB* on all data sets. Nevertheless, this approach is relying on the availability of annotated training data, which prevents ad-hoc queries. In comparison to the BART model that is used without fine-tuning on a given data set and set of query attributes (yellow bar), *WannaDB* achieves substantially better results. Especially for the Aviation and COVID19 data sets, *WannaDB* clearly outperforms this BART baseline. On the T-REx data sets, *WannaDB* provides competitive or better performance depending on the subset of data. We assume that BART's performance on the T-REx data is influenced by the fact that both the SQuAD data set it was fine-tuned on and the T-REx data set are based on Wikipedia.

**Generalization of BART.** As we have seen, while fine-tuning a BART model per data set yields the best performance, the BART model that is not fine-tuned for a data set provides inferior performance up to a point that it cannot extract any attributes correctly. To understand the generalization capabilities of BART in more depth and see if this is a systematic problem of BART, we now systematically use BART on data sets it has not been fine-tuned for. To be more precise, Figure 18.6b shows the results of two fine-tuned BART models: one fine-tuned on the *Aviation* data set and then used on the *T-REx Countries* data set and another model that we used vice versa; i.e., we applied both of them to the respective other data set, for which they have not been fine-tuned. The model fine-tuned on the *Aviation data* (reaching an F1 score of 91.95% tested in-domain on the *Aviation* data) only achieves 21.23% when tested on the *T-REx Countries* data set. At the same time, the model fine-tuned on the *T-REx Countries* data set (reaching an F1 score of $0.6633$ on the

[5] Used Checkpoint: *phiyodr/bart-large-finetuned-squad2* from Huggingface [Wol+19]

[6] In particular the fine-tuning task is QA on text collections which can be used to extract query attributes.

[7] The results of *WannaDB* and the second BART model that is used out-of-the-box are calculated on the whole data sets, whereas the results of the first BART model that is fine-tuned for the given data set are computed only on the 10% holdout test sets.

in-domain test set) fails completely for extracting information correctly from the unseen aviation data domain with an F1-score of $0.0$. This shows that a fine-tuned BART model is a valid approach to information extraction when annotated data is available and a fixed set of attributes is queried, but the resulting models are not able to generalize ad-hoc to other domains. In contrast, the results of *WannaDB* show that it can generalize well across data sets even without any particular training per data set and that the interactive approach provides an advantage over using generic embeddings or transformers directly.

**Detailed Analysis of *WannaDB*.** As a last point, we now zoom into the performance of *WannaDB* and analyze the results for all data sets on a per-attribute level to show that *WannaDB* can provide stable high performance and not just high performance for some query attributes. We used a combination of two different named entity recognizers,[8] Stanza [Qi+20] and SpaCy[9] [Hon+20] followed by our interactive matching approach.
Figure 18.7 shows that *WannaDB* can provide high accuracy and recall (measured by the combining F1 score, blue bars, right axis) for a wide spectrum of attributes from the three different data sets used in our evaluation. However, for some attributes the table is filled with a much lower quality than for others or not at all (e.g., for weather conditions). One reason can be that the currently employed information extractors are not able to extract the necessary information nuggets from the text (yellow bars). In particular, *aircraft_damage* and *weather_condition* are examples, where not only a large heterogeneity of mentions can be found but also very domain-specific terminology is used. Another reason for low table filling quality can be that the attributes occur in only a small fraction of the documents, as in the case of the attribute *owned_by* (which only occurs in 6% of the documents).

In conclusion, *WannaDB* has the advantage over fine-tuned BART models, that it neither requires annotated training data, nor several hours of training time in order to work on unseen text collections. Furthermore, it does not suffer from the problem of *hallucination* [May+20] that transformer-like models regularly experience, since they aim to also generate values for attributes even if no information nugget is present in the text. *WannaDB* instead generates an empty value in that case.

### 18.6.3. Exp. 3 – Effects of Interaction

In the previous experiments, we assumed a fixed amount of user interaction. In the third part of our evaluation, we instead investigate how the amount of interactive feedback given affects the table filling performance of *WannaDB*. We therefore simulate the interactive matching process with different interaction limits (i.e., the number of interactions per extracted query attribute). The resulting F1 scores can be seen in Figure 18.8.

As we can see, for some attributes, *WannaDB* achieves very high F1 scores with only one interaction with the user (e.g., for *event date* or *aircraft registration number* in the *Aviation* data set). These are attributes where the entity type of the extracted information nugget is very similar to the attribute name or the pattern of the extracted information nugget is rather unique. For example,

[8] *WannaDB* allows using multiple extractors at the same time, even if they produce overlapping nuggets. As default configuration for *WannaDB* and our experiments, we employ a combination of two robust general purpose extractors that are designed to work for a broad variety of domains. However, any other (combination of) extractors could be used in *WannaDB* as well.

[9] Using the *en_core_web_lg* model

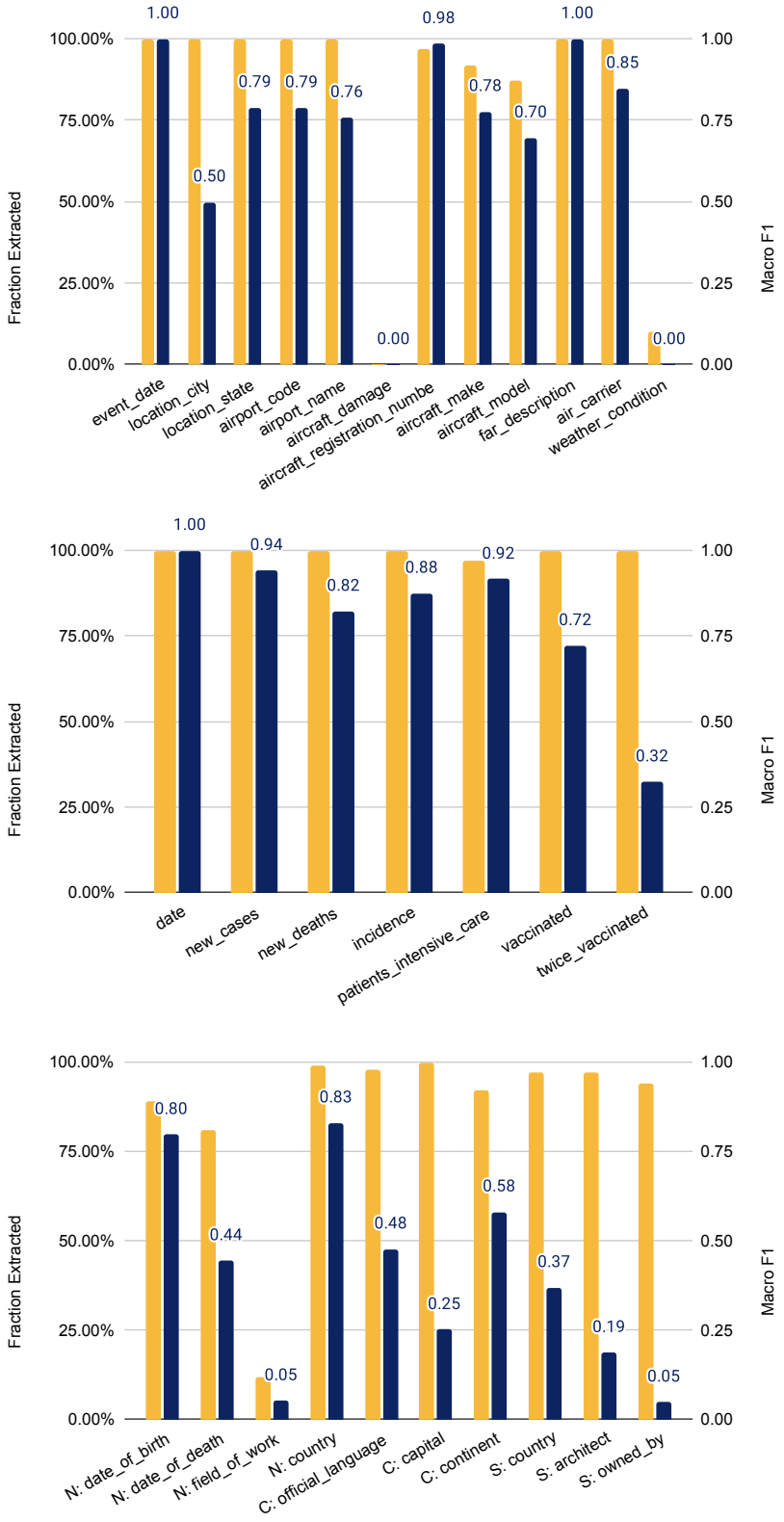



Figure 18.7.: ■ Fraction of values that could be extracted successfully and ■ table filling results per attribute of the Aviation, *COVID19* and T-REx data sets (in this order). *WannaDB* produces high scores for the majority of attributes, more than half are $0.7$ or above.

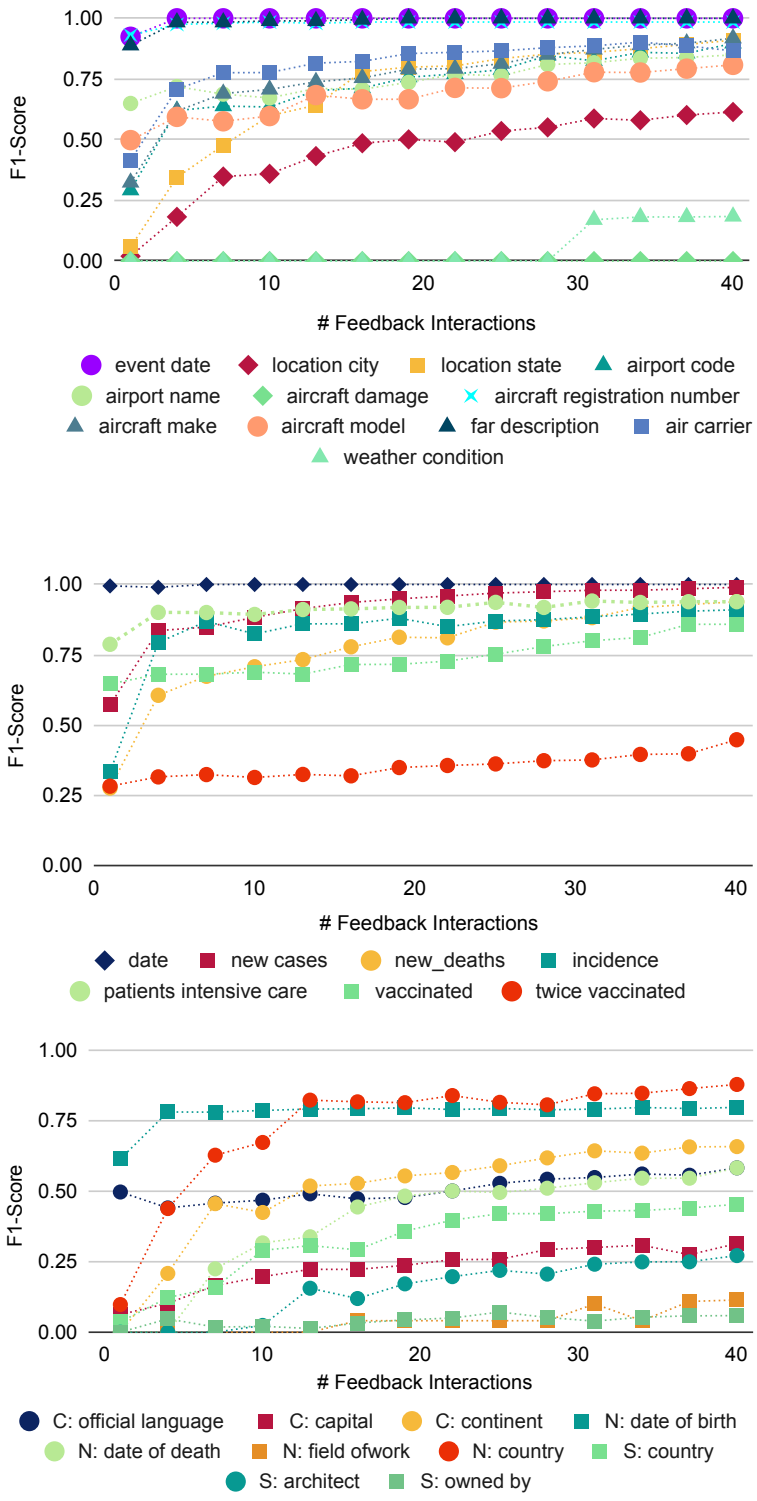



Figure 18.8.: F1 scores of *WannaDB* for the different attributes of the *Aviation*, *COVID19* and *T-REx* data sets for different amounts of feedback iterations per attribute (1-40). For most attributes, already a small amount of interactions drastically improves the quality, and more interactions lead to continuous improvements.

the extraction has the named entity tag *DATE* which is similar to *event date*. For other attributes though, the performance of *WannaDB* strongly depends on the amount of interactive feedback. However, important is that *WannaDB* can typically provide high quality with only a few interactions. For most attributes, the first $5 - 10$ interactions massively improve the F1-score to achieve gains of up to $0.5$. This overall confirms the interactive matching procedures we presented in Section 18.4 and the algorithm to select the right threshold. Yet, as we can additionally see, for a few attributes (e.g., weather condition), even many interactions cannot further improve the F1 scores. As we showed in the last experiment, the reason is that none of the extractors used in *WannaDB* can provide the information nugget for this attribute. Thus, as a future direction we want to combine *WannaDB* with a much broader set of existing extraction approaches beyond the named entity recognizers which we currently use, such as approaches for open information extraction.

### 18.6.4. Exp. 4 – Scalability

In our final experiment, we aim to assess the scalability of *WannaDB* to large text collections. Since *WannaDB* is an interactive system, the response times experienced by users are the most important performance metric. Across all used data sets, we measure that *WannaDB* takes on average $0.43$ seconds to process a single user interaction.[10] This latency includes all computations between two user interactions; i.e., updating the cached distances and guessed matches as well as presenting the next set of candidate matches to the user for feedback. In general, we find that the interaction latency scales linearly with the number of nuggets. To measure the offline extraction phase, which has to be executed only once per text collection, we report the runtime on our largest data set *T-REx Skyscrapers*, which comprises $2,683$ documents. Running our default extraction phase takes about $48$ minutes and produces $102,467$ nuggets. Comparing runtimes across data sets, we again find that the extraction runtime scales linearly with the number of generated nuggets.

In summary, it can be seen that *WannaDB* can scale to extensive text collections with thousands of documents and more than $100,000$ information nuggets by finishing the offline phase in a reasonable time and providing response times that allow for an interactive usage of the system [LH14].

## 18.7. Related Work

Running SQL queries on text collections is a new task, and to the best of our knowledge, there is no other system yet working in the same way as *WannaDB*. However, some parts of the task resemble existing tasks and for some components of our approach there is previous work. Therefore, in this section, we give an overview of the related work of different areas, including knowledge base population and schema matching based on embeddings.

**Information Extraction Systems.** Existing approaches to answer queries over text collections heavily rely on manual labor, requiring users either to read through vast amounts of texts and extract relevant information manually, or to build specific extraction pipelines. One category of information extraction systems focuses on the task of knowledge base population, where a

[10] We executed this and all other of our experiments on a consumer desktop machine (CPU: AMD Ryzen 9 3900X; RAM: 32GB @3000MHz; GPU: NVIDIA GeForce RTX 2070 SUPER with 8GB VRAM).

graph-structured knowledge base is constructed or expanded based on knowledge from natural language texts. Extractive approaches like DeepDive [Sa+16], SystemT [Chi+10], DefIE [BTN15], and QKBFly [Ngu+17] build upon (open) information extractors like ClausIE [CG13] and also perform the adaption, cleaning, and combination stages of the knowledge base building process. Most of these approaches require high manual efforts to design extraction pipelines for each knowledge base and domain specifically. Google Squared could be used to create fact-tables similar to the ones we propose from web contents, but was unfortunately discontinued without publications about the underlying techniques. Closest to our work are recent approaches for query-driven on-the-fly knowledge base construction, such as QKBFly. Yet, QKBFly extracts general subject-predicate-object triples and does not populate a user-defined table as *WannaDB* does. The vision of INODE [Ame+21] is to provide an end-to-end data exploration system that is also able to include information from natural language texts. For this task, the knowledge base population approach LILLIE [Smi+22] extracts triples from text domain-independently. However, the system has not been thoroughly evaluated for generalization to unseen domains. Recent approaches use transformer models to tackle information extraction tasks like relation extraction [CN21; EU21; Ngu+20] in an end-to-end fashion to avoid the errors accumulating in pipeline-based approaches. However, transformer-based methods are costly to train and suffer from issues like hallucination [May+20]. A more explainable approach to information extraction is introduced by [Kov+22; Rec+21] with a framework for learning text classifiers with a human-in-the-loop. Recently, [Sai+22] introduced an interactive system that allows users to specify templates that are then used to perform zero-shot information extraction.

**Text-To-Table.** The idea of automatically transforming a text into a table was also approached by [WZL22] as text-to-table task, which inversely tackles the well studied table-to-text problem. Yet, their work is not directly comparable, since they assume that each text fills one or more entire tables, while we assume that a text collection fills one table in which each text corresponds to a row.

**Template Filling & Named Entity Recognition.** The goal of slot or template filling is similar to our objective [GS96], yet in contrast to our approach, most template filling approaches are specifically crafted for a fixed set of slots. A common approach to extract a fixed set of attributes from a text is to learn a named entity recognizer specifically for the desired entity types (e.g., [SJ19]). Named entity recognizers extract a set of entity types like organizations, locations, or products from natural language texts. However, the training requires a substantial amount of annotated data, and the learned system will not generalize to entity types not present in the training data. Some approaches (e.g., [Che+15; Kho+17; Wei+19a]) attempt to avoid this problem by using active learning, which allows the learning algorithm to query the user, for example by selecting training instances that the user then labels by hand. Another strategy is distantly-supervised or weakly-supervised named entity recognition (e.g., [Fri+17; Lia+20]). In contrast to our system, these approaches train named entity recognizers specifically for the desired set of entity types, whereas we use the output of conventional named entity recognizers to populate the user-provided attributes. Together with the interactive matching, this allows *WannaDB* to generalize to unseen domains without the costly training of domain-specific named entity recognizers.

**Other Matching Tasks.** Approaches for schema matching (e.g., [Hät+20b; Her+20]), are related to *WannaDB*, too, since we frame the mapping between the information extractors' output and the user-provided list of attributes as a matching problem, but try to find correspondences between attributes and possible values, and not between columns or even full tables. Another recent approach focuses on matching texts to structured data, in particular also matching texts to table

rows [ASP21]. Yet, this task differs from the matching task in *WannaDB*, as it assumes the tables are given, whereas in *WannaDB* a table is filled through the matching.

**Entity Disambiguation & Cross Document Co-Reference Resolution.** The surface form of an entity in a text is often not sufficient to uniquely identify it. Yet, knowing whether two mentions of the same type describe the same entity is relevant for correct grouping in our case, but also existing tasks like entity linking/knowledge base alignment. For the latter there are three main challenges (see [Dre+10]): name variations (e.g., different mention forms, abbreviations, alternate spellings, and aliases), entity ambiguity (same written form for different entities), and absence (i.e., the text mentions a previously unknown entity). The last one is not relevant for our use-case, since we do not rely on a given KB but build tables only based on the current text collection. We can concentrate on the problem of ambiguity, i.e., decide, whether two nuggets that were matched as different rows of the same attribute are in fact the same or represent different concepts. The field of computing equivalence classes of textual mentions for the same entity is called cross-document co-reference resolution (CCR). It was, e.g., tackled by [Cat+21; DW15; KCP18], but these existing approaches often concentrate only on entities from certain domains or of certain types (like events).

**Prior Results of *WannaDB*.** A first version of the matching component of *WannaDB* including an initial evaluation on two real-world data sets was published at [HBB21]. In this paper, we pick up the vision of the whole application cycle presented at [Hät21]. As such, we present the integration of the table extraction procedure of *WannaDB* into a full system. Moreover, compared to the original submission, we also developed a new interactive matching procedure where we leverage the human ability to quickly find patterns by presenting multiple guessed matches at once, which allows users to quickly correct wrong matches. Multiple ways to give feedback (confirm, fix, or mark that there is no match in the document) further enhance quality and flexibility of matching. A demo of the interactive GUI for this matching process was presented at [HBB22].

## 18.8. Conclusions

In this paper, we presented *WannaDB*, a novel tool to explore the contents of unstructured data (text) using SQL-like queries in an ad-hoc fashion and without the need to manually design extraction pipelines upfront. It builds on embeddings and a novel interactive query execution strategy and consists of components to infer the required table structure from the query, extract and organize the required information from the text, group results on the embedding level and execute the query. Our evaluation shows that the individual components of *WannaDB* can achieve similar performance to models trained on large data sets for partial or related tasks, and gives an impression of the end-to-end quality that makes *WannaDB* suitable for many exploratory use cases.

# 19. WannaDB in Action: Deploying Ad-hoc SQL-over-Text Exploration in an Industrial Scenario (Under Submission)


## Abstract

*WannaDB* enables users to run SQL-like queries on extensive collections of textual documents. The approach works by interactively extracting and organizing domain-specific information without the need to manually specify extraction rules or provide large amounts of training data. In this paper, we showcase the real-world deployment of our tool in an industrial scenario at *Springer Nature* and conduct a multipart user study to prove its usefulness for real-world applications. Additionally, we present improvements to our query execution procedure compared to previous publications, as well as to the graphical user interface. Our code and documentation are open source.



## Bibliographical Information

The content of this chapter is currently under submission as "Benjamin Hättasch, Liane Vogel, Gard Jenset, Jan-Micha Bodensohn, Chandrima Roy, and Carsten Binnig. 'WannaDB in Action: Deploying Ad-hoc SQL-over-Text Exploration in an Industrial Scenario'. In: *Under submission* (2023)". The contributions of the author of this dissertation are summarized in Section 5.2.


## 19.1. Introduction

Having the right information at hand is crucial to making smart decisions, whether in research, engineering, finance, healthcare, or other fields. Accessing data is easier than ever before, as data from all areas, such as news, research papers, or government announcements, is published online and available around the clock through modern IT systems. Yet, distilling the relevant information from the sheer overwhelming amount of data is still an incredibly difficult task.

Academic publishing companies like *Springer Nature* have long since realized that simply providing access to papers and books is not enough to meet the needs of researchers, engineers, and other professionals. Instead, their focus has continuously shifted toward organizing information and giving the users new means to find and access precisely what they need. Research platforms like *SpringerMaterials*[1] provide enhanced and interactive visualizations, tabular overviews, and

[1] `https://materials.springer.com`

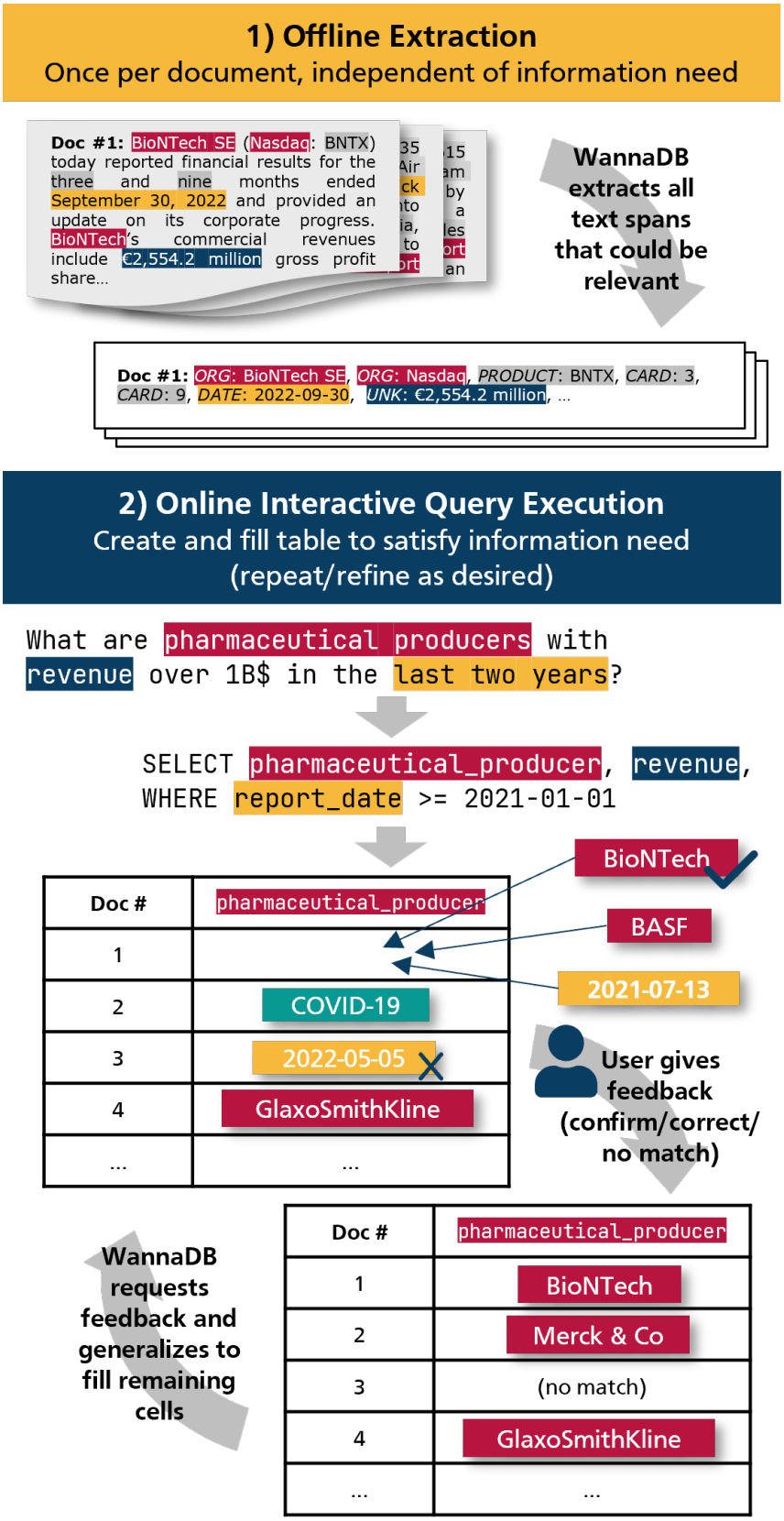



Figure 19.1.: Architecture & exemplary usage of *WannaDB*

domain-specific search functions backed by extensive and carefully curated databases. Additionally, tools and APIs for text and data mining support academic and industrial research.

Nevertheless, many data discovery techniques require technical background knowledge and are not easily accessible to domain experts from, for example, finance or healthcare. According to [KL10], there is a lack of easy to use tools for information extraction, as information overload can limit the performance of knowledge workers and not all existing tools provide sufficient benefit.

We therefore propose an approach that puts domain experts and knowledge workers in the driver's seat, acting as their own personalized data scientist. *WannaDB* allows them to interactively extract domain-specific information from text collections, organize it in tabular form, and find answers without requiring a trained data scientist. In this paper, we demonstrate how this can be helpful for tasks revolving around particular information needs, as well as the open-ended exploration of large text collections.

One example of such an extraction task could be financial news-gathering, where analysts must quickly obtain key figures like revenue, earnings per share, or mergers and acquisitions from company reports and press releases. Unfortunately, this task is cumbersome and error-prone when done by hand. Furthermore, off-the-shelf extraction systems often cannot extract all information the user is interested in. As domain experts and knowledge workers typically do not have the technical skills to craft custom extraction pipelines, they might experience a *context gap*, i.e., they are unable to bridge between their specific terminology and the labels used by existing extraction approaches. Moreover, knowledge workers are often not allowed to send sensitive data to third-party APIs, so one cannot always expect to have a domain-specific pre-trained language model at hand. Training a dedicated extraction model for such a task is often infeasible, since domain practitioners might lack the required training data. Finally, a learned model is in most cases not interpretable by humans, leading to a lack of trust in the results [Sch+22].

These problems are even more significant for the open-ended exploration of text collections, where the relevant attributes are not known upfront. Users need a data-first approach that allows them to quickly get insights without long feedback loops. Again, being able to quickly organize the data in tabular form can be of great help. Such a table may serve as the basis for further exploration and analysis, for example by revealing subsequent avenues of investigation, by providing initial answers through the computation of aggregates such as sums or averages, or by further filtering the document collection based on particular attribute values.

## 19.2. Overview of *WannaDB*

With *WannaDB*, we present such a tool that enables users to derive tabular representations from text collections and even run SQL-like queries on it. *WannaDB* extracts and organizes information by incorporating user feedback (see Figure 19.1). Instead of defining manual extraction rules (like, e.g., in [GB19]), users only need to confirm or rectify a small amount of guessed extractions in *WannaDB* (as can be seen in Figures 19.4 and 19.5).

Through user interaction, *WannaDB* gives users the opportunity to steer the extraction process and does not require large amounts of training data to adjust to new domains. Compared to trained black-box extraction models, *WannaDB* transparently shows the user what is extracted from the texts and gives them the opportunity to fix incorrect extractions right away. Approaches like this

are supported by a recent study [Sch+22], that emphasizes the importance of keeping the human in the loop to both increase efficiency and trust in tools and results. Moreover, this will get even more relevant in the future as regulations such as the *European General Data Protection Regulation* or the upcoming *AI Act* include a "right to explain."

The granularity of information that can be possibly extracted by *WannaDB* goes far beyond the one of classical named entity recognition and, in particular, incorporates the context in the target definition (e.g., finding the pharmaceutical company conducting a clinical trial as described in the current document vs. finding companies or even all kinds of organizations in that text). *WannaDB* even manages to do so for numeric values with different semantics, e.g., finding both the number of people vaccinated once and twice in a daily status report on the COVID-19 pandemic.

One could now argue that the task of information extraction from text could easily be addressed using large language models such as GPT-3 [Bro+20] or LLaMA [Tou+23]. However, their high resource requirements and long runtimes currently inhibit their use in exploratory settings, as we demonstrate even for smaller models such as BART [Lew+20]. In contrast, *WannaDB* requires only a one-time pre-processing per text collection on a single GPU, after which the text collection can be explored at minimal cost on typical consumer hardware (even on CPU-only machines). In addition, language models are often accessible only through an API and employ pay-per-query business models, where users are reluctant to openly explore the data as they have to pay for every single query. By contrast, *WannaDB* is released as an open-source tool that domain experts can apply directly to their own data.

Finally, our approach combines the advantages of language models or embeddings with the strict calculation possibilities and deterministic behavior of the SQL data query language. It can produce tables stating information that is not explicitly mentioned in the documents and hence not discoverable by pure extraction or search approaches, and can perform numerical reasoning on the data without the need to rely on the limited mathematical abilities [Hen+21] of a language model.

To allow for quick computation of query results on various domains, we build on two assumptions: First, *WannaDB* is currently limited to single-table queries on top of domain-specific document collections. In other words, we extract one table per document collection, where each row of the table corresponds to one document. However, this is not a severe limitation, as the extracted table can be seen as the materialized result of a join. For example, if a user needs information about a company's quarterly results along with general information about the company itself, *WannaDB* will extract a wide table containing all the necessary attributes for the query.

Second, the results produced by *WannaDB* are always approximate. There might be some incomplete or dirty values, but the approach allows creating tabular representations that contain the essence of a document collection with regard to certain user-specified attributes with only very little user feedback. In short, we focus on providing low-cost exploration document collections instead of exact results at much higher costs.

### 19.2.1. What does *WannaDB* provide?

*WannaDB* enables users to interactively extract domain-specific information and organize it in a table without the need to manually specify extraction rules or provide large amounts of training data. The basic idea is to first (independently of the user's information need) extract possibly relevant information snippets using off-the-shelf extraction systems, and then leverage a small

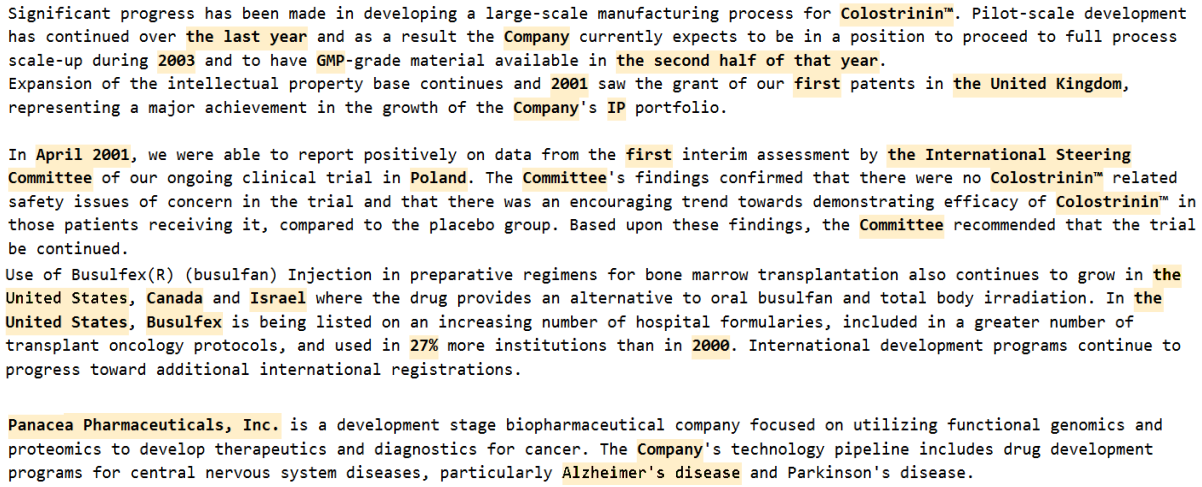



Figure 19.2.: Excerpts of press releases from the healthcare domain with highlighted extractions/nuggets from applying the first stage of *WannaDB*.

amount of user feedback to bridge the context gap and select the right information to fill a tabular representation with custom columns with information from the document collection.

The technical details and an extensive evaluation of our system were presented at [Hät+23a]. In this paper, we showcase the real-world deployment of our tool in an industrial scenario at *Springer Nature* and conduct a multipart user study to prove its usefulness for real world applications. Additionally, we present changes we made to the interactive table filling procedure to overcome previous shortcomings of our two-staged approach—allowing the user to select values that have not been extracted by the generic extraction systems in the first stage, too, and to leverage that information for the further table construction. Furthermore, we improved the graphical user interface initially presented at [HBB22] based on various feedback we received. We publish our code and documentation at `https://link.tuda.systems/wannadb`.

### 19.2.2. Lessons learned from the Industrial Deployment

In the remainder of this paper, we focus on the lessons learned from deploying *WannaDB* for an industrial use case at Springer. First, we explain the industrial use case in Section 19.3, followed by the details of how *WannaDB* works (Section 19.4). We then perform an evaluation, both with end-to-end experiments and a user study.

As part of this study, we conducted expert interviews, which underline the use of tools to extract and organize information without coding in exploratory or prototypic scenarios. The users liked the ability to quickly try many queries, and favored that our system adapts to their use of terminology and not vice versa. Our experiments show that our system works both substantially better and faster than a learned few-shot system that received the same amount of domain information from the user. Finally, our study confirms that real users can achieve a similar result quality as predicted in our simulations. More details on the evaluation can be found in Section 19.5.

Afterwards, we show similarities and distinctions of our system to existing approaches and tools in Section 19.6, before we explain how *WannaDB* can be used for your own applications in Section 19.7 and conclude in Section 19.8.

## 19.3. Industrial Scenario

In the following, we sketch the usage of *WannaDB* at *Springer Nature Data & Analytics Solutions*.[2] The description is based on a real scenario of a data science team that helps customers to answer data-driven questions. An evaluation and a user study based on such scenarios are shown in Section 19.5.

Typically, customers approach *Springer Nature* with a vague question like "Can we somehow get information on the current state of health equity from (publicly) available data?". What data could be used to answer that? A quick web search results in some statistics and reports by the *World Health Organization*, as well as many press releases from the healthcare sector. As a result, the data analyst working on the initial phase of this customer project ends up with a large collection of short texts from a lot of subtopics: some of the texts are focusing on certain diseases, others cover the medical situation in a certain area, some more describe clinical trials, etc.—examples are shown in Figure 19.2. The files are united by the fact that they probably all cover the health domain and therefore share a common terminology. Yet, they were written by different authors and published by multiple organizations, so structure, level of detail and in particular their focus will be different. In short, there are many diverse files, too many to read and understand in a reasonable time—especially when considering that the analyst still does not really know whether they really contain relevant information for them. Even categorizing them will already require a lot of effort, either for developing classification rules or labeling data to train a learned model.

This is the first place where an open exploration tool like *WannaDB* can help. An analyst stumbling through the dataset can use it to quickly check whether elements with the same meaning as they saw in a randomly opened document also appear in many others. Yet, this is not about finding exact words like it could be done with a keyword search, but instead checking, e.g., whether values representing the revenue of a pharmaceutical producer and the associated time span occur—without considering only fixed phrases like "revenue", "last year", or "Q4". The analyst will use *WannaDB* to quickly build an approximate table reporting these attributes for the document collection. For some texts, this will work well. For others, the cells will stay empty or contain values that are clearly unrelated. The analyst now knows that a subset of the texts deals with company key figures—but others probably cover other topics. They can (again led by intuition and some randomness) now repeat this process for other interesting attributes, like key numbers of clinical trials. *WannaDB* thus helps them to get an intuition regarding which topics are covered in the document collection. Furthermore, it can be used to divide the documents into these subtopics based on the presence or absence of attributes that the analyst perceives as typical for a particular category. All this time, the tool does not work with a fixed terminology, but instead based on the concepts that the analyst mentally connects to the terms they enter.

The analyst may now decide that they want to dive deeper into one of the areas touched by the data, e.g., have a closer look at certain diseases and regions. *WannaDB* can again help to quickly

[2] `https://www.springernature.com/gp/products/database/data-solutions`

and approximately extract and organize potentially relevant values, like counts and percentages of people affected by a disease or the regions the disease occurs in. The interactive approach again bridges between the mental model and the terminology used, and allows getting answers cheaply and without long waiting times. Furthermore, the SQL-like query structure enables filtering, ordering, and aggregation operations and combination of attributes.

They may now continue with other parts of the data and try to fill other target structures, or they may repeat the process on additional data like white papers to see whether they are suited to supplement this data. Overall, *WannaDB* allows analysts and other practitioners to explore a data collection, split it up, quickly get trends for some hypotheses, and learn whether the data is suitable for fulfilling certain information needs before spending a lot of time and money on extracting exactly the information required to transform it into a beneficial representation.

## 19.4. Interactive Query Execution

The architecture of *WannaDB* consists of two main stages, an offline extraction and pre-processing stage and an online interactive query execution stage. In this paper, we give a high-level overview of *WannaDB*'s system architecture. Further technical details can be found in our paper published at BTW'23 [Hät+23a].

**Extraction:** During the extraction stage, short information-bearing text snippets are greedily extracted from every document of the document collection using named entity recognizers (see Figure 19.1, upper part). As a default, we use off-the-shelf named entity recognizers from *Stanza* [Qi+20] and *spaCy* [Hon+20] as basic extraction systems. However, our system follows a framework approach here and could integrate any approach that identifies interesting text spans. In particular, this allows the usage of domain-specific resources that might be available in a company. The extracted values are then turned into information nuggets, which comprise embeddings for context information, in some cases a label, and the text span itself. These information nuggets are the basis for the interactive query execution, as *WannaDB* identifies which information nuggets contribute the most to answering the user's query with the help of user interaction.

The extraction stage needs to be run only once per document collection (independent of information need) and is fully parallelizable on the document level. The pre-processed extractions can easily be shared between users. In the industrial context, this means that the preprocessing can be done independently for every new document added to the exploration platform, be it uploaded by the user or from an automatic scraping or curating process—when additional documents are added to a text collection, the processing only needs to be done for the new documents without the need to re-process previous documents. This is an advantage of *WannaDB* over approaches where the latent embedding representation is dependent on all documents, and therefore has to be adapted for every document as soon as additional data is added.

**Interactive Query Execution:** At runtime, a user issues queries and interacts with *WannaDB* to execute them. The user can either enter an SQL-like query or specify a list of attributes they are interested in (see Figure 19.3). *WannaDB* automatically infers the table structure required to answer a query and employs a novel interactive table filling procedure to map the information

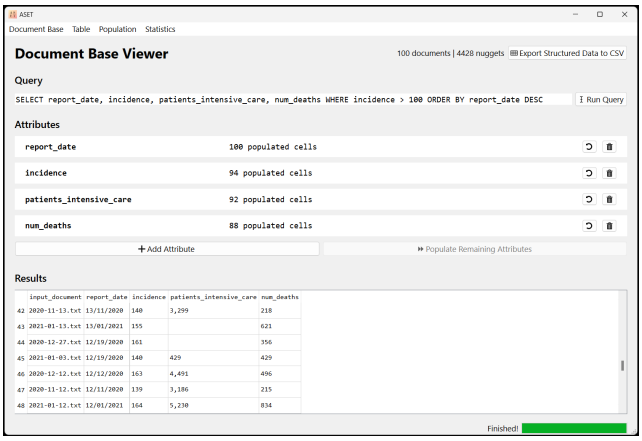



Figure 19.3.: Main interface of *WannaDB*, where the user can enter a query or manually specify attributes, manage the table's cell values, and inspect the results.

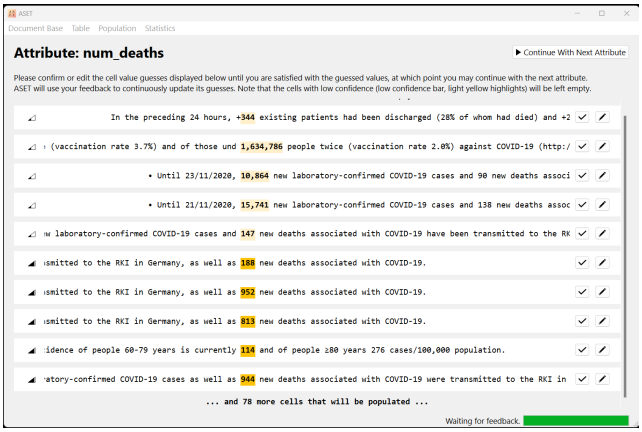



Figure 19.4.: Interactive table filling view that shows *WannaDB*'s currently guessed values for an attribute and allows the user to provide feedback.

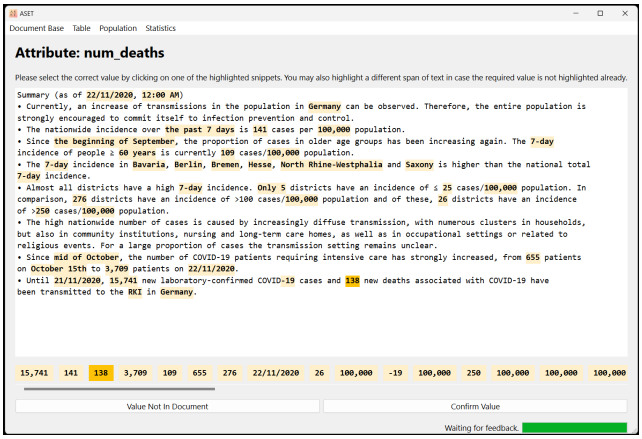



Figure 19.5.: Detailed inspection view for a particular document and attribute. The user can read the proposed table cell value in context, confirm it, adjust it by selecting another proposed extraction or a custom text span, or state that no value for this attribute occurs in the document. The list of proposed values at the bottom is ordered by the closeness to the next manually-confirmed value for this attribute.

nuggets extracted in the first stage to the required query attributes. To achieve that, *WannaDB* operates in the joint embedding space of information nuggets and target attributes (i.e., columns of the table), and decides which information from the document corresponding to the current table row is the correct value for the attribute and should be used to fill the cell based on cosine distances in the vector space. This provides transparency to the user, too: each automatically filled cell value will be chosen based on the closeness to a manually confirmed other value. Thus, our system does not only use the exploration of the vector space to fill the table, but that provides explanations for the behavior of the system as well. The user can inspect this information during the interaction phase and after the automatic filling of the remaining cells.

The interactive table filling works in an attribute-by-attribute fashion and by collecting user feedback: *WannaDB* displays a list of potential table cell values together with the contexts they appear in (see Figure 19.4). The user can select any of these rows and either confirm a cell value directly or switch to the edit and inspection view (see Figure 19.5) to carefully review the proposed value and either confirm it, select another proposed extraction or a custom text span, or state that the document at hand does not contain a value for that attribute. This feedback is then used to recompute distances between known and unknown values to update *WannaDB*'s guesses. The user continues to give feedback until they are satisfied with the quality achieved. This process is steered by carefully selecting which table value guesses are presented to the user for feedback to reach a high table filling quality with as little feedback as possible. At the same time, since *WannaDB* always presents multiple rows to the user at once, our approach can leverage the human ability to quickly identify (incorrect) entries that stand out without the need to carefully read all rows the system is currently unsure about.

To allow for empty cells in case a document does not state the required value, *WannaDB* applies a distance threshold and only populates a table cell with its best-fitting value if this value's distance in the embedding space is lower than the threshold. This threshold is automatically tuned based on the user's feedback to account for variances in the closeness of vectors for different attributes. By using only a single threshold per attribute, we treat the vector space uniformly, even though the "closeness level" in a vector space will probably never completely correlate with the semantic closeness. Our experiments, however, show that this threshold is normally enough to decide which values should not be considered at all (i.e., which ones are so "off" that it is better to rather keep a cell empty). By tuning only a single threshold per attribute, we can achieve that kind of generalization with a minimal number of feedback interactions.

In addition to previously published versions of our approach, we now make it possible to manually select arbitrary relevant text spans at runtime in case they were not (or only partially) extracted by the greedy extraction stage. *WannaDB* will then use the manually-selected span to fill the cell for the corresponding document. Furthermore, this new nugget is added to the embedding space, and marked as a confirmed element there, such that existing nuggets from other documents close to it can be selected as candidates by the system. As a further way to profit from this user feedback, *WannaDB* will also scan other documents for potentially relevant text spans that are similar to this manually-selected one and add them to the vector space to select the nuggets from. That way, we overcome some of the issues of our strict two-staged approach without losing its advantages (i.e., the fast response times during the interactive query execution without the need for a powerful machine).

## 19.5. Evaluation on Real-World Data

The main focus of the evaluation in this paper is placed on the industrial usage of our tool, which includes a multipart user study on a real-world scenario at Springer Nature. Additionally, we show the end-to-end performance of *WannaDB* on real-world datasets and compare both the result quality and the runtime to a learned few-shot approach using a similar amount of feedback. For an in-depth technical evaluation of the individual components of our system, the scalability, as well as a comparison with multiple further baselines, we refer to our paper published at BTW'23 [Hät+23a].

### 19.5.1. End-to-End Performance

To measure the end-to-end performance of *WannaDB*, we simulate a fixed amount of user feedback (20 interactions per attribute) for a list of target table columns on each text collection and measure the resulting performance (F1-score[3]) for each of these attributes. We compare these scores to a few-shot training/refining approach that received the same amount of input/feedback, i.e., a BART sequence-to-sequence model fine-tuned on 20 labeled examples. For both approaches, we additionally measure the time needed to construct the full table, hence for our tool the time to evaluate and incorporate the user feedback and for the few-shot approach the time needed for training and prediction.

**Few-shot Baseline:** To put our results in perspective, we compare *WannaDB* with a few-shot approach trained on the same number of labeled examples per attribute that is provided to *WannaDB* in the evaluation setting: we fine-tune a BART [Lew+20] model from the Huggingface [Wol+19] library on 20 randomly[4] picked datapoints for each attribute of each dataset, simulating the scenario of replacing the internal workings of *WannaDB* by a learned model. The information extraction task is thereby formulated as a sequence-to-sequence task (i.e., the input is a text document and the output is the structured data extracted from the text). Each model is trained for up to 20 epochs, with a learning rate of $1e-5$. We split the 20 datapoints in 15 for training and 5 for validation and select the best checkpoint based on the performance on the validation set.

**Datasets:** For this evaluation, we use three datasets: *Aviation*, a text collection based on $100$ aviation accident reports, *COVID-19*, again $100$ documents describing the situation of the Covid-19 pandemic in Germany and focusing on numeric attributes with a lot of cases where the system should deliberately keep the resulting cell empty, and *T-REx*, an adapted version of the T-REx dataset [ElS+18] which is based on Wikipedia articles with aligned Wikidata triples from which we extracted three subsets (about Nobel laureates, countries, and skyscrapers). More details on

[3] We thereby consider cells (i.e., an attribute value) as true positives when they are correctly filled with information from the text corresponding to that row, and as true negatives when they are correctly left empty, in case the required information is not present in the corresponding text. False positive predictions occur when a cell is filled incorrectly. False negatives occur when a cell is left empty that should have been filled with data from the text, and also for incorrectly filled cells, as the correct nugget has not been found.

[4] In contrast, one core functionality of *WannaDB* is to leverage the feedback to propose cells to give feedback to. This is not possible here, since the few-shot approach requires all examples at once for fine-tuning and validation. Adapting that would largely increase the runtime.

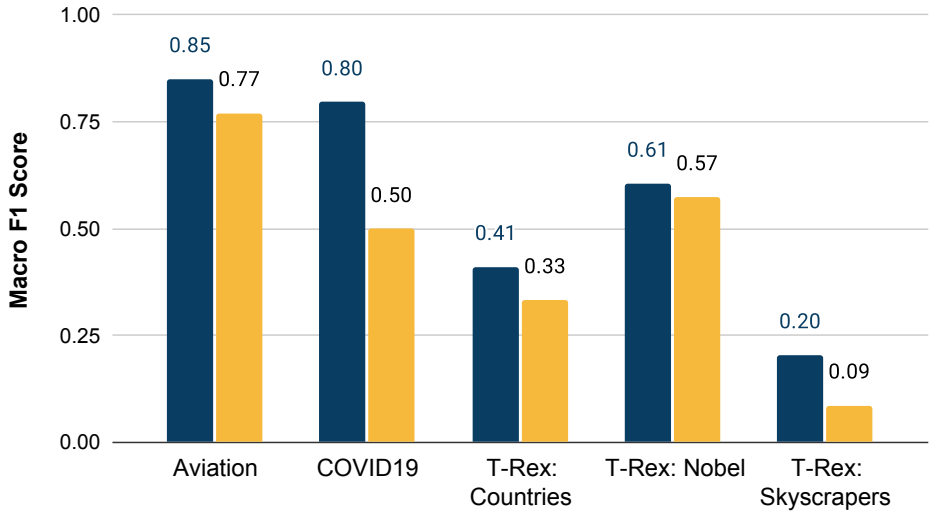



Figure 19.6.: Table filling results. It can be seen that ■ *WannaDB* outperforms the ■ few-shot BART model (fine-tuned on 20 datapoints) on every dataset.

the datasets can be found in [Hät+23a]. For each of these datasets, we either manually annotated the correct values for each attribute and document (Aviation, COVID-19) or used the existing alignment (T-REx).

**Table Filling:** The results of *WannaDB* in comparison to the few-shot BART baseline for table filling are visualized in Figure 19.6. It can be seen that *WannaDB* consistently outperforms the few-shot baseline approach on all datasets. The highest performance difference is notable on the COVID-19 dataset, where *WannaDB* reaches an F1 score of $80\,\%$, whereas the BART baseline models only reach $50\,\%$. We suspect that this is partly due to the limited capabilities of language models such as BART when working with numerical values [Hen+21], as there are many numerical attributes contained in the COVID-19 dataset, such as the incidence or the number of patients in intensive care. Additionally, we noticed that the BART baseline models have difficulties when many of the texts do not mention the required values and thus many cells need to be left empty. Meanwhile, *WannaDB* shows consistently good performance extracting numerical attributes, as well as having fewer problems when dealing with missing data. An intuition that our system indeed explores the vector space and leverages closeness can be found in Figure 19.7. It shows a low-dimensional projection of the guesses of the system after different numbers of feedback iterations compared to the gold standard shown in the same way. It can be seen that *WannaDB* quickly converges to the selection of the correct values for nearly all attributes of the dataset.

**Runtime Analysis:** As *WannaDB* is an interactive tool, it is critical to achieve a short latency when processing user interactions and incorporating the feedback to populate the table cells, since too much waiting time for users makes a system uncomfortable to use. We therefore measure the time needed for our system to process 20 user interactions per attribute and apply the feedback to automatically fill the remaining cells. For *WannaDB*, we report the time on a machine with a CPU only (AMD Ryzen 9 3900X; RAM: 32GB @3000MHz) as well as on one equipped with a

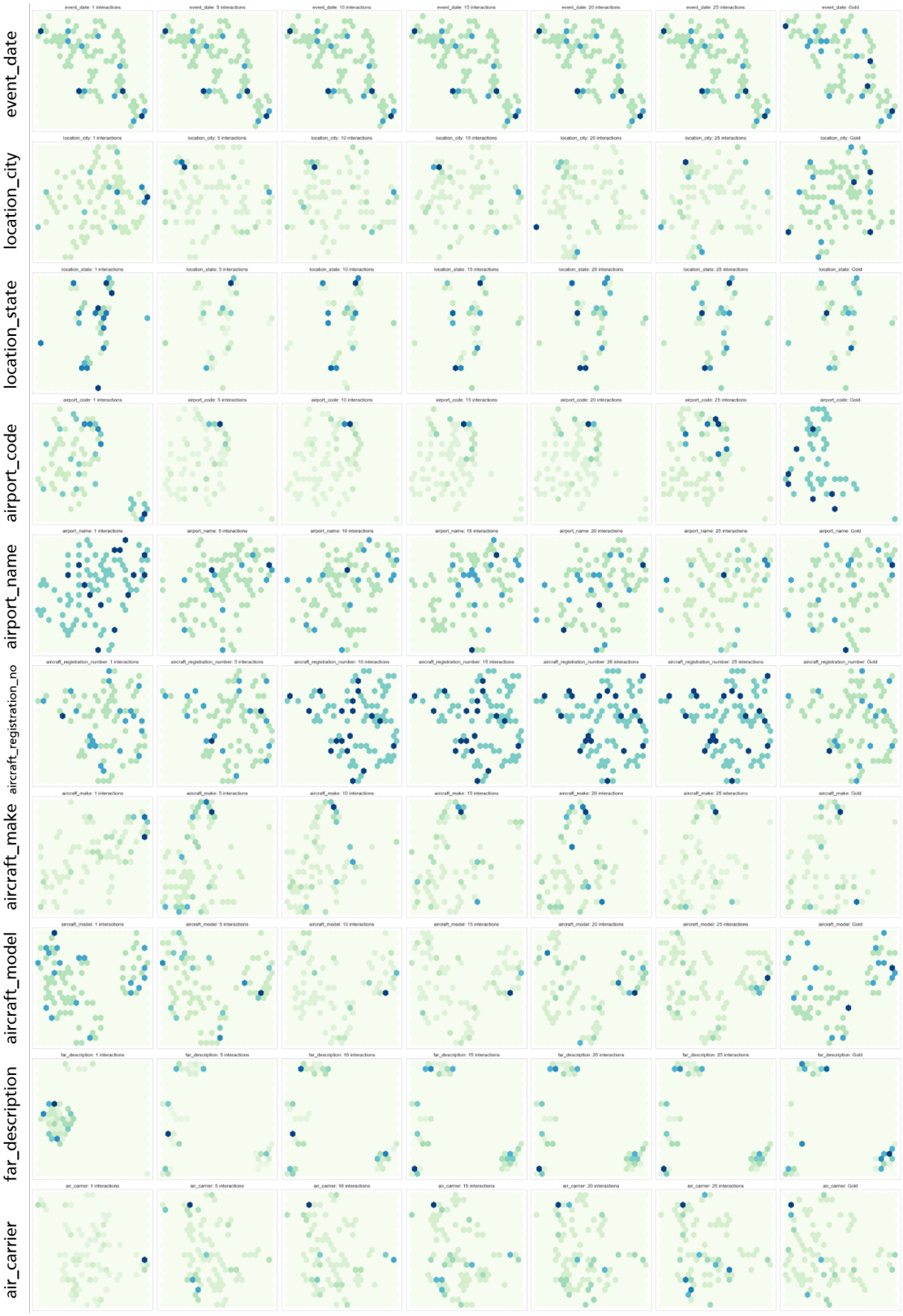



Figure 19.7.: Visualization of the explored vector space for the aviation dataset. The important dimensions for each attribute are highlighted using UMAP and plotted as a hexbin visualization, showing the guesses of *WannaDB* after 1, 5, 10, 15, 20, and 25 interactions compared to the gold selection in the last column.

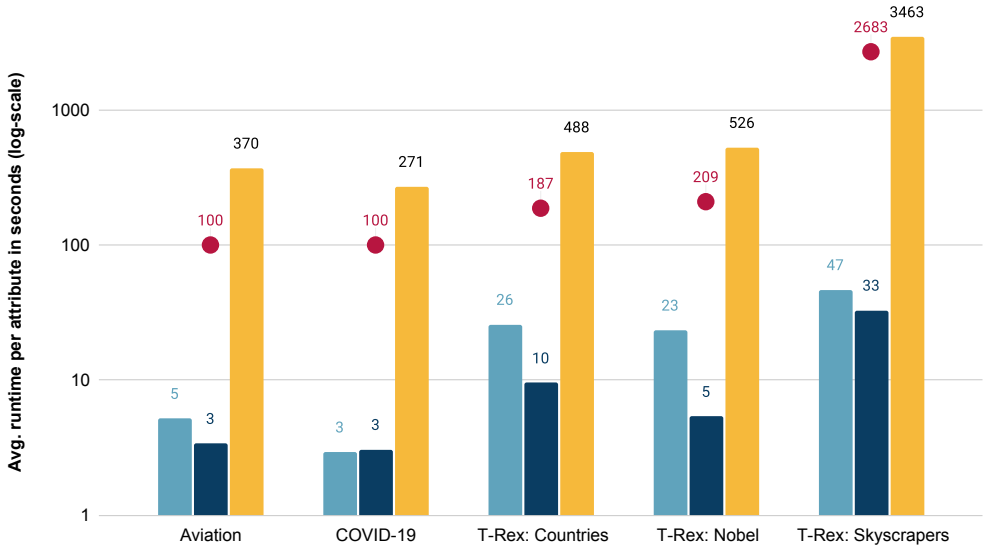



Figure 19.8.: Average runtime per attribute in seconds - logarithmic scale. We compare ■ *WannaDB* on a CPU and ■ *WannaDB* on a consumer GPU with a ■ few-shot BART model on a server GPU. The ■ dots show the number of documents per dataset on the same logarithmic scale. It can be seen that running ■■ *WannaDB* takes orders of magnitudes less time than the ■ BART model.

GPU (NVIDIA GeForce RTX 2070 SUPER with 8GB VRAM). For the few-shot approach, we only report the performance for training and prediction on a machine with a GPU designed for machine learning (NVIDIA A100-SXM4-40GB). Using a consumer machine without an ML-optimized GPU, this approach would take at least five hours per attribute, rendering it completely unusable for exploratory scenarios.

Figure 19.8 visualizes the results of our runtime measurements; we report the average over 20 runs. For better readability, we chose a logarithmic scale and visualize the average time per attribute in seconds for each of our datasets, together with the respective dataset's size. It can be seen that running *WannaDB* takes orders of magnitudes less time compared to the language model-based BART approach. On our largest dataset *T-Rex: Skyscrapers* (2683 documents), *WannaDB* needs around $47$ seconds total computation time for all calculations together (not including the time waiting for the user to perform feedback actions) on a machine with a CPU only, whereas the BART model nearly takes an hour to predict the cell values from all the 2683 documents, even though it is running on a much more powerful machine. Even when incorporating the preprocessing time, which only has to be done once for all attributes (our default extraction phase takes about $48$ minutes and produces $102,467$ nuggets for the *T-Rex: Skyscrapers* dataset) and only use a consumer GPU for that, our approach will be faster than a run for a single attribute with the BART model on an ML-optimized machine. Overall, our runtime analysis shows that *WannaDB* is indeed usable in an interactive fashion and does not cause long waiting times for the user as a system based on a language model like BART would. It can also be seen that our system scales well with the number of documents.

**Summary:** In conclusion, this evaluation shows that *WannaDB* achieves continuously better performance on all our datasets while at the same time being orders of magnitude faster than a pre-trained language model such as BART. Additionally, *WannaDB* has several other advantages over the baseline, such as being more transparent and interpretable (i.e., through the notion of closeness between automatically selected and user-confirmed values) and not suffering from the problem of hallucination that transformer-based models regularly experience [May+20], since they aim to also generate values for attributes even if the information is not present in the text. *WannaDB* instead leaves the corresponding cell empty.

### 19.5.2. User Study in the Industrial Scenario

*WannaDB* is an interactive tool that is intended to assist real-world users. Therefore, we also want to evaluate the users' experience of working with the system. Both the core algorithms including the means of feedback and the graphical user interface were iteratively refined over the last years [Hät21; HBB21; HBB22; Hät+23a]. In this study, we now want to systematically evaluate *WannaDB*: in the first part, we want to learn about the expectations of data analysts, data scientists and knowledge workers for such a tool and how well it integrates with their existing tool chains and pipelines. In the second part of this study, we will focus on evaluating how intuitively users can use the tool and how good the table filling quality is with real user feedback, as our previous evaluations were based on simulated user feedback.

#### Method

We carried out our study with ten people overall. In the first part, we conducted two expert interviews with senior data analysts working at *Springer Nature* in the areas of dataset curation and visualization. The interviews considered the field of work of the participants, their relation to working with unstructured data, and other tools they are currently using. The participants were asked to assess the tool's basic ideas and its application areas. To evaluate the usefulness of *WannaDB* for them in a real word exploratory use case, this was combined with an open exploration phase of our prototype with think-aloud reporting on data already familiar to them from their current work.

For the second part of our study, we recruited eight PhD students with a computer science background. The users were given a short written introduction to *WannaDB* and a short explanation of the scenario of extracting information from a text collection of status reports about the Covid-19 situation from the German RKI. Afterwards, they were asked to perform 7 tasks in a predefined order. During those tasks, they received further information in the form of a short, written manual explaining some backgrounds of the system and describing possible actions using some screenshots. As the final part of the study, the users were asked to fill out a questionnaire including both rating and free text questions.

#### 1st Part – Insights and Results

The first interviewee needs to transform unstructured data into some kind of tabular representation as part of his daily work routines, a task that currently consists of many "fairly manual" steps

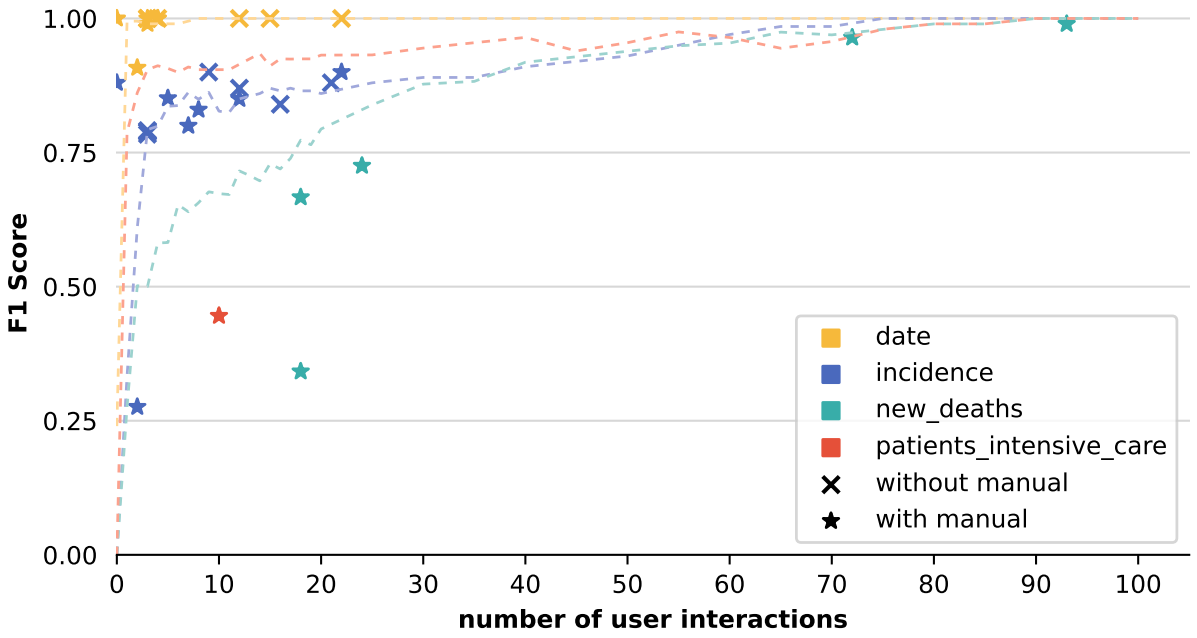



Figure 19.9.: User study table filling results. We visualize the table filling F1-score dependent on the amount of feedback a user gave for each attribute, for the first attempt without the manual ($\times$) and the second attempt after reading the manual ($\star$). The lines indicate the table filling quality achieved by the simulated user feedback. It can be seen that even with a small number of interactions ($< 10$), a high quality can be achieved. Experience with the system (reading the manual before and using it for the second time) mostly decreased the number of interactions required to achieve a similar quality. The performance by the participants is mostly similar to the one of a simulated user.

and often requires "quite complex coding." For him, a useful exploration tool for textual contents should "reduce the time spent on coding" and "on initial data collection" and "allow to quickly and easily create tables." He sees a strong chance of time saving when data can be transformed "without the need to manually compose extraction pipelines." When using *WannaDB*, he assumes that people might need some time and training to come up with useful attributes, but likes the possibility to quickly confirm extractions in an *appealing interface* ("seems to be useful to save time"). Additionally, he is delighted that the tool performs generalization/adaption without him needing to care about the machine learning processes.

The second interviewee works together with data experts on the one hand, and visualization and UX experts on the other. As part of his tasks, he needs to learn *what kind of information exists in datasets to create suitable visualizations from that*. His main tools are spreadsheets, running SQL on databases and some rapid prototype coding. The process of developing visualizations often requires multiple rounds of iterative improvements, where he has to wait to receive suitable data and integrate it to then test new ideas and submit them for feedback. He would *profit from the ability to quickly generate approximate data* while working together. Another important task for him is *filtering large datasets* to build *smaller datasets covering subtopics*, which can be a prerequisite

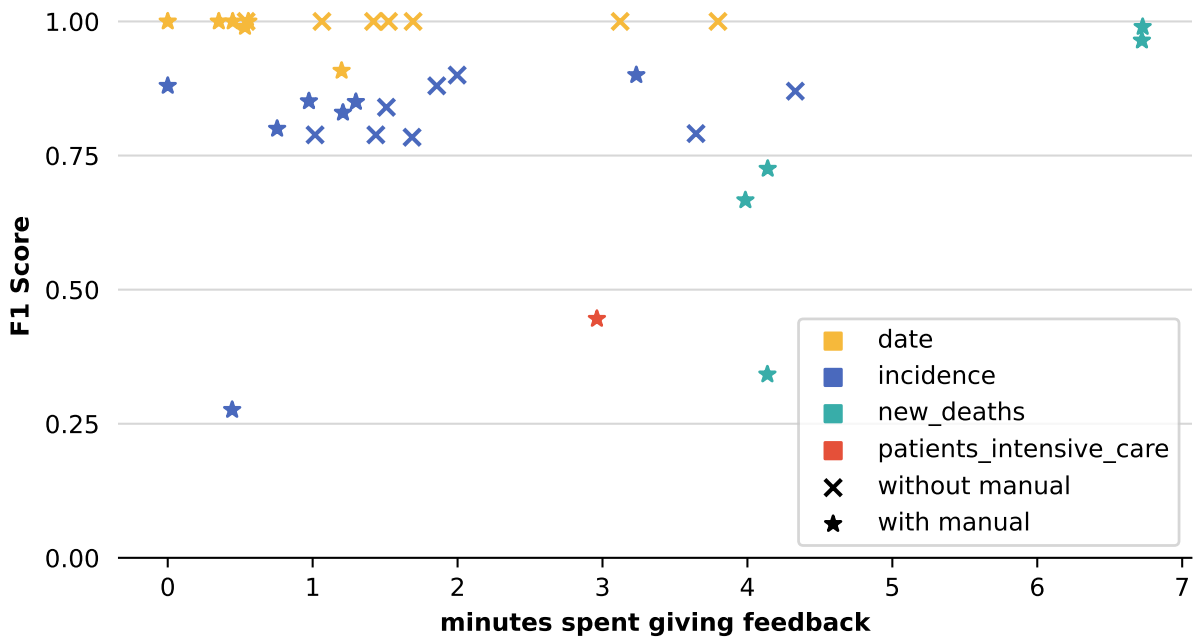



Figure 19.10.: User study table filling results. We visualize the table filling F1-score dependent on the minutes a user spent giving feedback for the attribute, for the first attempt without the manual ($\times$) and the second attempt after reading the manual ($\star$). After reading the manual and being more familiar with the tool, in the second attempt users needed only about half the time to reach a comparable quality.

for several analytic and visualization steps. Finally, he emphasizes the fact that a *simplified version of a dataset*, e.g., "a bar chart instead of a map" can often be the best way to convey a message.

Presented with the idea of the tool, he likes the idea since it "implies some kind of automation, that automatically delivers you a result without heavy lifting." He imagines that it can be used to "run multiple queries, compare the results, and decide which, e.g., two out of five queries are the ones you should continue with." When using *WannaDB* for the first time, he needs some hints ("I'm not sure whether I need to enter a specific name of a field of an existing dataset or whether I can give it an arbitrary name like 'Bob'") but then understands how he can specify the target structure he is interested in. He also correctly infers from the interface how selecting a nugget or custom text span are used to update the table of current guesses during the interactive table population. After trying the tool, he points out that this "looks like it could help people to independently work on data without needing to ask people to do this for you." He imagines *literacy review* to be one field of application, and compliments on the "filtering possibilies beyond [those in] Excel."

In summary, both interviews confirm the intended application and the need for such a tool to reduce manual processing steps in exploring datasets. Both participants commented positively on the user experience and are interested in learning more about the tool in the future.

### 2nd Part – Tasks

The second part of the study used a list of tasks that were handed out to the participants: First (Task 1), users were asked to open 2-3 of the documents in order to get a first impression of the data they will be working with. Next (Task 2), they were asked to load the document collection into *WannaDB*, and to enter the attributes *date* and *incidence* (Task 3). *WannaDB* provides multiple options to enter attributes, and we left open which option to choose, observing which way seemed most convenient to the users. Afterwards, users were asked to start populating the table cells by interacting with *WannaDB*. They were given no additional hints regarding the best strategies for giving feedback apart from the guidance provided by the user interface. They could decide on their own when to stop giving feedback based on whether they were satisfied with the reached extraction quality. Only after finishing the cell population on their own, they were given a manual to read through (Task 4), explaining more background on the internal workings of *WannaDB*. After reading through the manual, we again asked the users to populate the table cells for the attributes *date* and *incidence* (Task 5). Here, we wanted to measure how their strategies for giving feedback change after knowing more about *WannaDB*, and whether that impacts the overall quality of the extracted results. As a next task (Task 6), users should further explore *WannaDB* by entering another attribute they were interested in. They were also allowed to look again at a few documents to come up with another attribute to extract. As a last task (Task 7), users were asked to export the extracted data to a `csv` file. Finally, users were sent a questionnaire to rate the usability of *WannaDB* using the System Usability Scale (SUS) [Bro95] and to answer additional questions, as well as to provide feedback on what they liked about the tool and what improvements they would suggest.

### 2nd Part – Results

Overall, all participants in our user study were successful in completing the seven tasks. The initial loading of the documents at the start (Task 2) and the exporting of the result table at the end (Task 7) were performed without difficulties by all users, as they all were able to quickly detect the functionality in the tool.

**Table Filling Performance:** For the tasks regarding the extraction of the attributes *date* and *incidence* from the documents, we visualize the result quality in Figure 19.9 and 19.10. In Task 2, users were asked to fill the result table for the two attributes without having background knowledge on the internal workings of *WannaDB*. The F1-score dependent on the number of feedback interactions for each attribute is visualized using $\times$-shaped markers in Figure 19.9. It can be seen that for the attributes *date* and *incidence*, users gave feedback between $\sim 5 - 25$ times, and were able to reach F1-scores higher than $0.75\,\%$. In Figure 19.10 it can be seen that it took users between $1$ and $4.5$ minutes to give feedback for each of the two attributes in this first attempt, before they achieved an extraction quality they were satisfied with. The results of the second attempt of filling the table (Task 5, after reading the manual), are visualized in both Figures with $\star$-shaped markers. It shows that in the second attempt, both the number of interactions (only up to $5$ for the attribute *date*) and the time needed were reduced, while reaching extraction qualities that are comparable to the first attempt. Additionally, we noticed during the study that the participants now decided more deliberately which kind of feedback to give based on the state of the

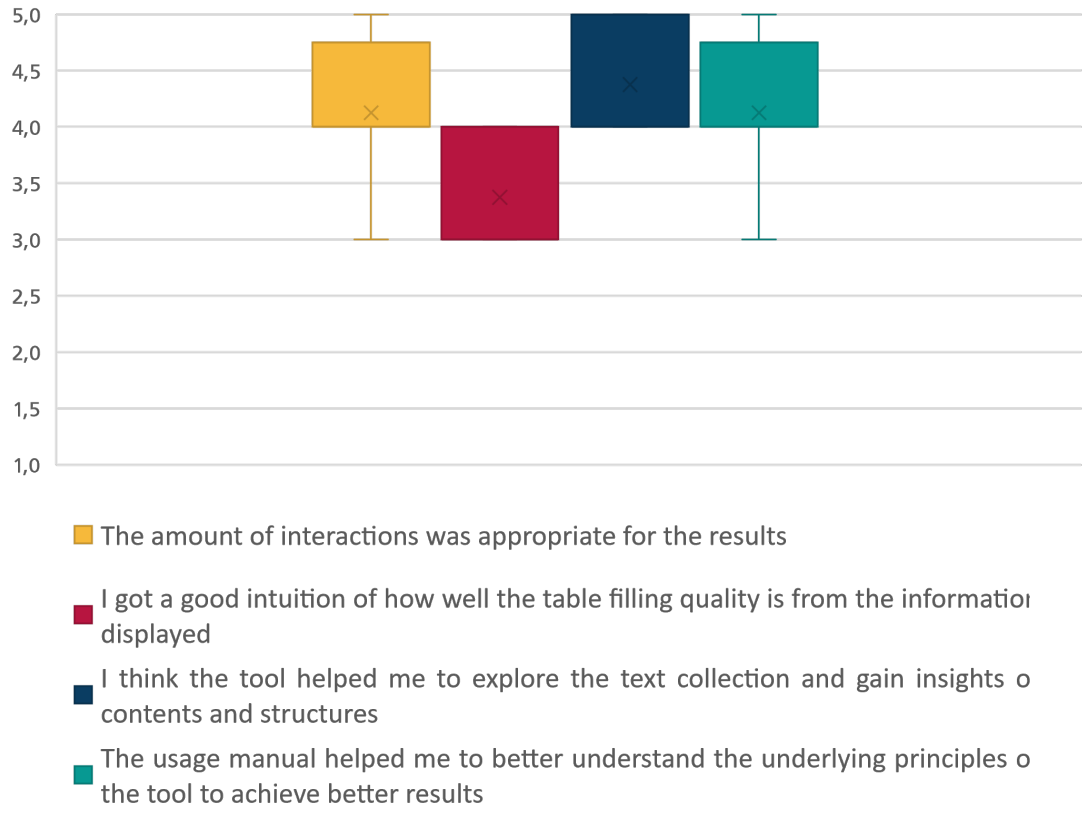



Figure 19.11.: Answers of the participants to rating questions in the post-survey questionnaire. Scale from 1 – Strongly Disagree to 5 – Strongly Agree.

currently displayed entries. We find that our system allows for a steep learning curve, where some background information from the manual, as well as being a bit more familiar with the tool in the second attempt, helps to reduce the needed interaction time quickly. The resulting performance values are similar to the ones when simulating the user feedback for the same amount of iterations (assuming perfect feedback for randomly chosen entries). This affirms the results presented in [Hät+23a].

**Exploration Observations:** In order to explore the text collection on their own and to get further insights into the capabilities of *WannaDB*, users were also asked to add attributes they came up with on their own and extract them from the documents with the help of our tool (Task 6). The users chose the attributes *deaths, intensive care, death percentage, state*, and *high incidence districts*. The attribute *deaths* was chosen independently by 5 out of 8 users. The users were free to decide how much feedback they want to provide for extracting the attribute and when to stop the table filling process. This lead to a wider range of results regarding the reached F1-scores for the custom chosen attributes, as some users were more ambitious than others to reach a very good quality and were therefore willing to provide more feedback.

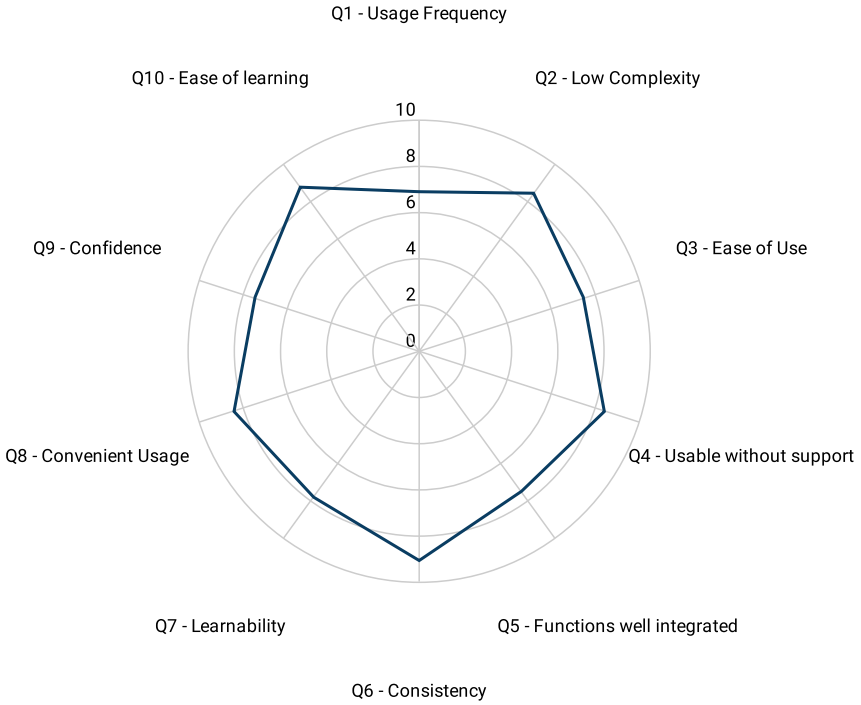



Figure 19.12.: User scores for the 10 questions from the System Usability Scale (SUS) [Bro95] (higher scores are better).

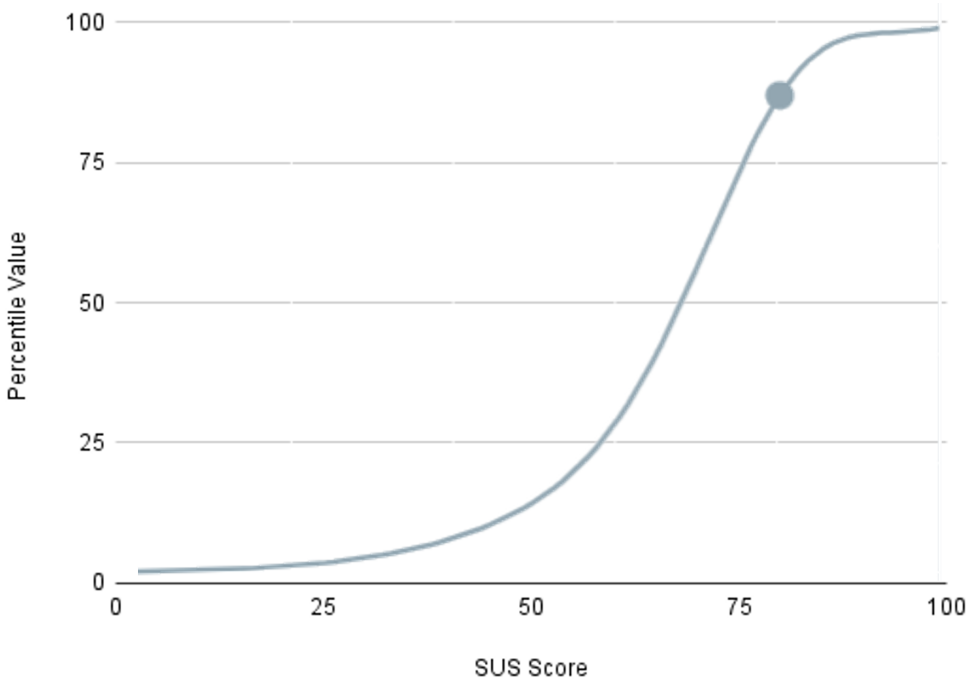



Figure 19.13.: *WannaDB* achieves a SUS score of $80.31$, which is part of the $85 - 89$ percentile, corresponding to a grade of A-.

**System Usability Scale:** After completing the seven tasks, the participants were asked to fill out a questionnaire containing the ten questions of the system usability scale (SUS) [Bro95]. We visualize the average result per question (normalized to values between 0 and 10) in Figure 19.12. Overall, *WannaDB* receives a SUS score of $80.31$, which is part of the $85 - 89$ percentile (see Figure 19.13) and corresponds to the grade *A-*, the rating *acceptable* on the acceptability scale, and *good* on the adjective rating scale developed by [BKM09].

**Comments and Questionnaire Results:** Furthermore, in the questionnaire, the participants were asked to give a degree of agreement for some questions regarding the interaction with the tool, the usefulness and the manual. The results can be found in Figure 19.11; overall the ratings are positive.

Asked about potential use cases of *WannaDB*, participants imagined, e.g., a usage as "preparation step to quickly generate plots over textual data", to "verify whether certain information is sufficiently present in a source" or to "evaluate free text answers in feedback surveys." The participants reported that they, e.g., liked the *gamification aspects* of optimizing the displayed scores through feedback actions, "that there was no restrictions of what attributes/columns to extract", allowing for extraction of any imaginable attribute, the *general idea of the iterative approach and generalization*, the *quick reactions after feedback* and *the possibility to both confirm and correct entries*, and "how quickly the tool learned to extract values from text with only a few interactions."

Think-aloud reporting during the usage of the tool led to ideas for improvements (see next section) and other valuable insights:

The users liked that they could select a proposed extraction by simply clicking on it. We could notice that different participants followed very different strategies for feedback, some always tried to feedback the first or last entry, some preferred one type of feedback (confirming or correcting entries), some tried to actively balance between them. Moreover, some assumed a sufficient quality once no or even only a few wrong cells were visible in the table population view, while others continued confirming even after they noticed some convergence for a certain time "to be sure." This may also depend on the reason to do such a task and whether there is an intrinsic interest in the result.

We were also reminded that one cannot expect that all users carefully read instructional texts in the software or a manual (by questions about topics explicitly stated there). However, it was mentioned by multiple participants that they rely on short explanations and tooltip texts and prefer them over just icons or very short labels.

Especially for attributes that require some more rounds of feedback, we could quickly notice that the participants soon started detecting patterns, and then explicitly checked for them, e.g., by looking at certain "typical" positions in the document when correcting an entry. Finally, participants reported that exactly those attributes require more feedback to correctly fill than others, where more simulated feedback was needed in our previous evaluations, too.

### Suggested Improvements

We asked all participants for suggestions on how we could further improve the tool and got valuable ideas that we plan to integrate in *WannaDB*. In particular, we will further improve the way the

difference between cells filled with high confidence and those that will not be considered for the result tables (distance threshold) will be visualized, since multiple participants needed explanations from the manual to perceive the difference. Users suggested adding the ability to scroll through extractions from more documents than only the 10 extractions that are currently presented by *WannaDB* (Figure 19.4). Also suggested was a search functionality when inspecting a document (Figure 19.5) and the ability to undo the last action, for example in cases where an extraction has been confirmed by mistake. We also received other minor suggestions like adding a link from the result table to the raw document a row was extracted from. Additionally, users suggested updates to the provided manual that we will incorporate, and we also got further ideas on how to improve the user interface to make it even more intuitive.

Overall, both parts of our study confirmed the usefulness for the planned application. We got very positive feedback regarding most aspects of the existing user interface, the approach itself, and the quality of the results.

## 19.6. Distinction from existing systems

Running SQL queries on text collections is a new task, and to the best of our knowledge, there is no other system yet with the same capabilities as *WannaDB*. However, some parts of the task resemble existing tasks and for some components of our approach there is previous work. Therefore, in this section, we give an overview of the related work of different areas, as well as explain the distinction of *WannaDB* from existing systems or approaches.

**Information Extraction Systems:** Existing approaches to answer queries over text collections heavily rely on manual labor, requiring users to either read through vast amounts of text and extract relevant information manually, or to build specific extraction pipelines.
One category of information extraction systems focuses on the task of knowledge base population, where a graph-structured knowledge base is constructed or expanded based on knowledge from natural language texts. One possibility to build them are manual annotations, possibly supported by a tool that displays knowledge already gathered graphically (e.g., [SGL08] provides a visual analytic system to explore connections between entities found in document collections). Extractive approaches like DeepDive [Sa+16], SystemT [Chi+10], DefIE [BTN15], and QKBFly [Ngu+17] build upon (open) information extractors like ClauseIE [CG13] and also perform the adaption, cleaning and combination stages of the knowledge base building process. However, these approaches require high manual efforts to design extraction pipelines for each knowledge base and domain specifically, whereas *WannaDB* generalizes to unseen domains and information needs.
Another line of work explores the generation and application of information extraction rules based on user specified examples [Han+17; Sch+22]. Google Squared could be used to create fact-tables similar to the ones we propose from web contents, but was unfortunately discontinued without publications about the underlying techniques. Another approach for fact table creation is Jigsaw [GB19], with a focus on scalable processing of large document collections. While they also work with document level nuggets that are organized to fill the table rows, matches between them and target attributes are not automatically inferred, but have to be specified with complex hand-crafted extraction rules.

**Named Entity Recognition:** *WannaDB* builds upon extractions from existing named entity recognition (NER) systems [Hon+20; Qi+20]. While named entity recognition is a sub-field of information extraction, there are several reasons why it is not sufficient on its own for answering SQL queries over text collections. Most importantly, training NER systems requires a substantial amount of annotated data, and the learned system will not generalize to entity types not present in the training data. Training domain-specific NER systems for information extraction is necessary, as the extraction categories of most existing NER systems are not fine-grained enough to answer specific information needs, e.g., *companies* and *universities* are often both tagged as *organization*. To overcome this limitation, *WannaDB* employs a novel interactive table population approach to match information from texts to the user's query, without the costly training of domain-specific named entity recognizers.

**Supervised Learning Approaches:** For the ad-hoc exploration of text collections, supervised learning approaches are not easy to apply due to the lack of labeled training data. Building a sufficiently large corpus of annotated training data is costly because, on the one hand, a very wide range of possible domains has to be covered and, on the other hand, the SQL queries must be varied depending on the user's information need, which is difficult to realistically simulate when creating a dataset.

**Large Language Models:** Very recently, large language models such as GPT-3 [Bro+20] with billions of parameters have shown remarkable performance also for information extraction tasks. However, there are several ways, in which *WannaDB* is better suitable for the exploration of text collections, than large language models are: while the pre-processing of text collections with *WannaDB* is a one time effort and afterwards users can explore the collection freely, large language models need to process all texts of a text collection again whenever the user poses a new query. Due to the size of language models, this takes time and requires substantial computing resources. Currently, large language models provided over APIs have restrictive limits or are only available through pay-per-query offers, which restricts the open-ended exploration of text collections as every query leads to additional costs. To avoid pay-per-query costs, running a local copy of a large language model is only an alternative when sufficient computing resources are available. In comparison, *WannaDB* only requires a machine with a consumer GPU for pre-processing once for each new text-collection, the exploration works also on a typical consumer computer without using a GPU.

## 19.7. Ready to Use

*WannaDB* is not only a novel approach for information extraction and organizing, but a ready-to-use tool both for individual usage and industrial deployment, covering preprocessing, query interpretation, interactive matching and computation of the final results.

We created a graphical user interface for our tool, that is meant to be used by people regardless of their background. In contrast to the first publication about this interactive tool [HBB22], we iteratively improved the UI and added new capabilities like direct input and execution of SQL-like queries.

Additionally, we provide a stand-alone preprocessing script that can be used to perform the more computing-heavy preprocessing step on a stronger server equipped with a GPU and then load that preprocessed version of the document collection for the interactive matching using the GUI on a normal consumer machine.

*WannaDB* is dually licensed under both an Open Source license for the free usage by end users or the embedding in Open Source projects, and a commercial license for the integration in industrial projects and closed-source tool chains. Code, documentation and further materials can be found at `https://link.tuda.systems/wannadb`.

## 19.8. Conclusion & Future Work

In this paper, we showcased the usage of our tool in an industrial scenario from the healthcare domain and conducted an extensive user study to prove its usefulness for real world applications. Additionally, we presented improvements we applied to our query execution procedure compared to previous publications, as well as further enhancements to the graphical user interface.

In our evaluation, we showed that *WannaDB* outperforms a few-shot baseline both with regard to quality and runtime. Experts confirmed the suitability for the intended application in an industrial scenario. Moreover, the participants of our user study rated the usability and intuitiveness of our tool as high and were able to complete all tasks with a sufficient result quality in an adequate amount of time. The results also confirmed the evaluation results using simulated users shown in our previous publications.

In the future, we want to further leverage user inputs by generalizing manually added extractions to additional extraction rules at runtime. Additionally, we plan to improve our software by adding more export formats and creating a browser-based version of the user interface. We will consider the feedback received in the study to make sure our software is intuitive and easy to use.

# Acronyms

**AI** artificial intelligence

**CA** conversational agent

**CPU** central processing unit

**CS** computer science

**DBMS** database management system

**ILP** integer linear program

**IT** information technology

**LLM** large language model

**LM** language model

**MDS** multi-document summary

**NL** natural language

**NL2SQL** natural language to SQL

**NLI** natural language interface

**NLIDB** natural language interface for databases

**NLP** natural language processing

**NLU** natural language understanding

**OLTP** online transaction processing

**SPARQL** SPARQL Protocol and RDF Query Language

**SQL** Structured Query Language

# Bibliography


[AR22] Daron Acemoglu and Pascual Restrepo. “Demographics and Automation”. In: *The Review of Economic Studies* 89.1 (Jan. 1, 2022), pages 1–44. ISSN: 0034-6527. DOI: 10.1093/restud/rdab031. (Visited on 10/27/2023).

[ASP21] Naser Ahmadi, Hansjorg Sand, and Paolo Papotti. “Unsupervised Matching of Data and Text”. In: *CoRR* abs/2112.08776 (2021). arXiv: 2112.08776.

[Ame+21] Sihem Amer-Yahia, Georgia Koutrika, Martin Braschler, Diego Calvanese, Davide Lanti, Hendrik Lücke-Tieke, Alessandro Mosca, Tarcisio Mendes de Farias, Dimitris Papadopoulos, Yogendra Patil, Guillem Rull, Ellery Smith, Dimitrios Skoutas, Srividya Subramanian, and Kurt Stockinger. “INODE: Building an End-to-End Data Exploration System in Practice”. In: *SIGMOD Rec.* 50.4 (2021), pages 23–29. DOI: 10.1145/3516431.3516436.

[Ana21] Anaconda. *Anaconda | State of Data Science 2021*. Anaconda. 2021. URL: https://www.anaconda.com/resources/whitepapers/state-of-data-science-2021 (visited on 10/28/2023).

[And+95] Ion Androutsopoulos et al. “Natural language interfaces to databases - an introduction”. In: *Natural Language Engineering* 1.1 (1995), pages 29–81.

[ALM16] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. “A simple but tough-to-beat baseline for sentence embeddings”. In: *The International Conference on Learning Representations (ICLR)* (2016).

[Aro+23] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. *Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes*. Apr. 20, 2023. DOI: 10.48550/arXiv.2304.09433. arXiv: 2304.09433 [cs]. preprint.

[Avi+18] Avinesh P. V. S., Benjamin Hättasch, Orkan Özyurt, Carsten Binnig, and Christian M. Meyer. “Sherlock: A System for Interactive Summarization of Large Text Collections”. In: *Proc. VLDB Endow.* 11.12 (2018), pages 1902–1905. ISSN: 2150-8097. DOI: 10.14778/3229863.3236220.

[AM17] Avinesh P. V. S. and Christian M. Meyer. “Joint Optimization of User-desired Content in Multi-document Summaries by Learning from User Feedback”. In: *ACL*. ACL, 2017, pages 1353–1363.

[APM18] Avinesh P. V. S., Maxime Peyrard, and Christian M. Meyer. “Live Blog Corpus for Summarization”. In: *Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC)*. Miyazaki, Japan, May 2018, pages 3197–3203. URL: http://www.lrec-conf.org/proceedings/lrec2018/pdf/317.pdf.

[BCB15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *ICLR*. Volume abs/1409.0473. 2015. arXiv: 1409.0473.

[Ban+23] Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, Quyet V. Do, Yan Xu, and Pascale Fung. *A Multitask, Multilingual, Multimodal Evaluation of ChatGPT on Reasoning, Hallucination, and Interactivity*. Feb. 28, 2023. DOI: 10.48550/arXiv.2302.04023. arXiv: 2302.04023 [cs]. preprint.

[BKM09] Aaron Bangor, Philip Kortum, and James Miller. “Determining what individual SUS scores mean: Adding an adjective rating scale”. In: *Journal of usability studies* 4.3 (2009), pages 114–123.

[Bar+15] Marco Bartolozzi, Pierfrancesco Bellini, Paolo Nesi, Gianni Pantaleo, and Luca Santi. “A Smart Decision Support System for Smart City”. In: *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*. 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity). Dec. 2015, pages 117–122. DOI: 10.1109/SmartCity.2015.57. URL: https://ieeexplore.ieee.org/abstract/document/7463711?casa_token=FcArx_C69vcAAAAA:bKBopcSd9tICCTIGOq6WmwTHdvXBDRAS6FiumJ38C-6t9Xt7pBq6bm4kNJEHg-6d5EUWQIMhytE (visited on 10/26/2023).

[Bas+18] Fuat Basik, Benjamin Hättasch, Amir Ilkhechi, Arif Usta, Shekar Ramaswamy, Prasetya Utama, Nathaniel Weir, Carsten Binnig, and Ugur Çetintemel. “DBPal: A Learned NL-Interface for Databases”. In: *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*. Edited by Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein. ACM, 2018, pages 1765–1768. DOI: 10.1145/3183713.3193562.

[BEM14] Islam Beltagy, Katrin Erk, and Raymond Mooney. “Semantic Parsing using Distributional Semantics and Probabilistic Logic”. In: *ACL 2014 Workshop on Semantic Parsing*. 2014, pages 7–11.

[BL14] Jonathan Berant and Percy Liang. “Semantic Parsing via Paraphrasing”. In: *ACL*. 2014, pages 1415–1425.

[Ber+13] Sonia Bergamaschi, Francesco Guerra, Matteo Interlandi, Raquel Trillo Lado, and Yannis Velegrakis. “QUEST: A Keyword Search System for Relational Data based on Semantic and Machine Learning Techniques”. In: *PVLDB* 6.12 (2013), pages 1222–1225.

[BH13] Rahul Bhagat and Eduard H. Hovy. “What Is a Paraphrase?” In: *Computational Linguistics* 39.3 (2013), pages 463–472.

[Blu+12] Lukas Blunschi, Claudio Jossen, Donald Kossmann, Magdalini Mori, and Kurt Stockinger. “SODA: Generating SQL for Business Users”. In: *PVLDB* 5.10 (2012), pages 932–943.

[BMF15] Florian Boudin, Hugo Mougard, and Benoit Favre. “Concept-based Summarization using Integer Linear Programming: From Concept Pruning to Multiple Optimal Solutions”. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMLNP)*. Lisbon, Portugal, 2015, pages 1914–1918. ISBN: 978-1-941643-32-7. URL: http://aclweb.org/anthology/D15-1220.

[BTN15] Claudio Delli Bovi, Luca Telesca, and Roberto Navigli. “Large-Scale Information Extraction from Textual Definitions through Deep Syntactic and Semantic Analysis”. In: *Trans. Assoc. Comput. Linguistics* 3 (2015), pages 529–543. DOI: 10.1162/tacl_a_00156.

[Bri99] Sergey Brin. “Extracting Patterns and Relations from the World Wide Web”. In: *The World Wide Web and Databases*. Edited by Paolo Atzeni, Alberto Mendelzon, and Giansalvatore Mecca. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1999, pages 172–183. ISBN: 978-3-540-48909-2. DOI: 10.1007/10704656_11.

[Bro95] John Brooke. “SUS: A quick and dirty usability scale”. In: *Usability Eval. Ind.* 189 (Nov. 1995).

[Bro+20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. “Language Models are Few-Shot Learners”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[BP19] Tim vor der Brück and Marc Pouly. “Text Similarity Estimation Based on Word Embeddings and Matrix Norms for Targeted Marketing”. In: June 2019, pages 1827–1836. DOI: 10.18653/v1/N19-1181.

[Bur23] Bureau of Labor Statistics, U.S. Department of Labor. *Data Scientists : Occupational Outlook Handbook: : U.S. Bureau of Labor Statistics*. 2023. URL: https://www.bls.gov/ooh/math/data-scientists.htm (visited on 10/26/2023).

[CN21] Pere-Lluís Huguet Cabot and Roberto Navigli. “REBEL: Relation Extraction By End-to-end Language generation”. In: *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*. Association for Computational Linguistics, 2021, pages 2370–2381. DOI: 10.18653/v1/2021.findings-emnlp.204.

[Caf+07] Michael Cafarella, Christopher Ré, Dan Suciu, and Oren Etzioni. “Structured Querying of Web Text Data: A Technical Challenge.” In: CIDR. Jan. 1, 2007, pages 225–234.

[Cai+18] Ruichu Cai, Boyan Xu, Zhenjie Zhang, Xiaoyan Yang, Zijian Li, and Zhihao Liang. “An Encoder-Decoder Framework Translating Natural Language to Database Queries”. In: *IJCAI*. 2018, pages 3977–3983.

[Cat+21] Arie Cattan, Sophie Johnson, Daniel S. Weld, Ido Dagan, Iz Beltagy, Doug Downey, and Tom Hope. “SciCo: Hierarchical Cross-Document Coreference for Scientific Concepts”. In: *3rd Conference on Automated Knowledge Base Construction, AKBC 2021, Virtual, October 4-8, 2021*. 2021. DOI: 10.24432/C5B594.

[Cer+18] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. "Universal Sentence Encoder". In: *CoRR* abs/1803.11175 (2018). arXiv: 1803.11175.

[Che+19] Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles Sutton. "Learning semantic annotations for tabular data". In: *arXiv preprint arXiv:1906.00781* (2019).

[Che+12] Nengcheng Chen, Jie He, Chao Yang, and Chao Wang. "A node semantic similarity schema-matching method for multi-version Web Coverage Service retrieval". In: *International Journal of Geographical Information Science* 26.6 (2012), pages 1051–1072.

[CC18] Xilun Chen and Claire Cardie. *Unsupervised Multilingual Word Embeddings*. Sept. 6, 2018. DOI: 10.48550/arXiv.1808.08933. arXiv: 1808.08933 [cs]. preprint.

[Che+15] Yukun Chen, Thomas A. Lasko, Qiaozhu Mei, Joshua C. Denny, and Hua Xu. "A study of active learning methods for named entity recognition in clinical text". In: *J. Biomed. Informatics* 58 (2015), pages 11–18. DOI: 10.1016/j.jbi.2015.09.010.

[Chi+10] Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, and Shivakumar Vaithyanathan. "SystemT: An Algebraic Approach to Declarative Information Extraction". In: *ACL 2010, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, July 11-16, 2010, Uppsala, Sweden*. The Association for Computer Linguistics, 2010, pages 128–137. URL: https://aclanthology.org/P10-1014/.

[Chr+14] Janara Christensen, Stephen Soderland, Gagan Bansal, and Mausam. "Hierarchical Summarization: Scaling Up Multi-Document Summarization". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, June 2014, pages 902–912. DOI: 10.3115/v1/P14-1085. URL: https://www.aclweb.org/anthology/P14-1085.

[Chu+17] Shumo Chu, Chenglong Wang, Konstantin Weitz, and Alvin Cheung. "Cosette: An Automated Prover for SQL". In: *CIDR*. 2017.

[CV07] Rudi L Cilibrasi and Paul MB Vitanyi. "The google similarity distance". In: *IEEE Transactions on knowledge and data engineering* 19.3 (2007), pages 370–383.

[CC04] Stephen Clark and James R. Curran. "Parsing the WSJ Using CCG and Log-Linear Models". In: *ACL*. 2004, pages 103–110.

[Con+17] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. "Supervised Learning of Universal Sentence Representations from Natural Language Inference Data". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*. Copenhagen, Denmark: Association for Computational Linguistics, 2017, pages 670–680. DOI: 10.18653/v1/d17-1070.

[CG13] Luciano Del Corro and Rainer Gemulla. “ClausIE: clause-based open information extraction”. In: *22nd International World Wide Web Conference, WWW ’13, Rio de Janeiro, Brazil, May 13-17, 2013*. International World Wide Web Conferences Steering Committee / ACM, 2013, pages 355–366. DOI: 10.1145/2488388.2488420.

[Cra+00] Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew McCallum, Tom M. Mitchell, Kamal Nigam, and Seán Slattery. “Learning to construct knowledge bases from the World Wide Web”. In: *Artif. Intell.* 118.1-2 (2000), pages 69–113.

[Cro+15a] Andrew Crotty et al. “Vizdom: Interactive Analytics through Pen and Touch”. In: *PVLDB* 8.12 (2015), pages 2024–2035.

[Cro+15b] Andrew Crotty, Alex Galakatos, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. “Vizdom: interactive analytics through pen and touch”. In: *PVLDB* 8.12 (2015), pages 2024–2027.

[Cro+16] Andrew Crotty, Alex Galakatos, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. “The case for interactive data exploration accelerators (IDEAs)”. In: *HILDA@SIGMOD*. ACM. 2016, page 11.

[Dao+19] Tri Dao, Albert Gu, Alexander Ratner, Virginia Smith, Chris De Sa, and Christopher Ré. “A Kernel Theory of Modern Data Augmentation”. In: *ICML*. Volume 97. 2019, pages 1528–1537.

[DP22] Thomas H. Davenport and D. J. Patil. “Is Data Scientist Still the Sexiest Job of the 21st Century?” In: *Harvard Business Review* (July 15, 2022). ISSN: 0017-8012. URL: https://hbr.org/2022/07/is-data-scientist-still-the-sexiest-job-of-the-21st-century (visited on 10/26/2023).

[Deh+17] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W. Bruce Croft. “Neural Ranking Models with Weak Supervision”. In: *SIGIR*. 2017, pages 65–74.

[Dev+18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “BERT: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[Dev+19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, 2019, pages 4171–4186. DOI: 10.18653/v1/n19-1423.

[Dia23] Hussein Dia. *Electric Vehicle Fires Are Very Rare. The Risk for Petrol and Diesel Vehicles Is at Least 20 Times Higher*. The Conversation. Sept. 15, 2023. URL: http://theconversation.com/electric-vehicle-fires-are-very-rare-the-risk-for-petrol-and-diesel-vehicles-is-at-least-20-times-higher-213468 (visited on 10/27/2023).

[Dil19] Ben Dilday. *TF-IDF for tabular data featurization and classification*. June 6, 2019. URL: https://medium.com/enigma-engineering/tf-idf-for-tabular-data-featurization-classification-4cd5b034ce62 (visited on 08/10/2019).

[DR02] Hong Hai Do and Erhard Rahm. "COMA - A System for Flexible Combination of Schema Matching Approaches". In: *Proceedings of 28th International Conference on Very Large Data Bases, VLDB 2002, Hong Kong, August 20-23, 2002*. Morgan Kaufmann, 2002, pages 610–621. DOI: 10.1016/B978-155860869-6/50060-3. URL: http://www.vldb.org/conf/2002/S17P03.pdf.

[Do05] Hong-Hai Do. "Schema Matching and Mapping-based Data Integration". PhD thesis. Germany: Interdisciplinary Center for Bioinformatics and Department of Computer Science, University of Leipzig, 2005.

[DL16] Li Dong and Mirella Lapata. "Language to Logical Form with Neural Attention". In: *ACL*. Volume abs/1601.01280. 2016.

[DL18] Li Dong and Mirella Lapata. "Coarse-to-Fine Decoding for Neural Semantic Parsing". In: *ACL*. 2018, pages 731–742.

[Don+23] Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, lu Chen, Jinshu Lin, and Dongfang Lou. *C3: Zero-shot Text-to-SQL with ChatGPT*. July 14, 2023. DOI: 10.48550/arXiv.2307.07306. arXiv: 2307.07306 [cs]. preprint.

[Dre+10] Mark Dredze, Paul McNamee, Delip Rao, Adam Gerber, and Tim Finin. "Entity Disambiguation for Knowledge Base Population". In: *COLING 2010, 23rd International Conference on Computational Linguistics, Proceedings of the Conference, 23-27 August 2010, Beijing, China*. Tsinghua University Press, 2010, pages 277–285. URL: https://aclanthology.org/C10-1032/.

[DUC06] *Document Understanding Conference 2006 Corpus*. http://duc.nist.gov/duc2006. Accessed: 2018-03-01. 2006.

[DUC07] *Document Understanding Conference 2007 Corpus*. Accessed: 2018-03-01. 2007. URL: https://www-nlpir.nist.gov/projects/duc/data/2007_data.html.

[DR13] Martin Dürr and Klaus Radermacher. *Einsatz von Datenbanksystemen: ein Leitfaden für die Praxis*. Springer-Verlag, 2013.

[DW15] Sourav Dutta and Gerhard Weikum. "Cross-Document Co-Reference Resolution using Sample-Based Clustering with Knowledge Enrichment". In: *Trans. Assoc. Comput. Linguistics* 3 (2015), pages 15–28. DOI: 10.1162/tacl_a_00119.

[EU21] Markus Eberts and Adrian Ulges. "An End-to-end Model for Entity-level Relation Extraction using Multi-instance Learning". In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*. Association for Computational Linguistics, 2021, pages 3650–3660. DOI: 10.18653/v1/2021.eacl-main.319.

[Ebr+18] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. "Distributed representations of tuples for entity resolution". In: *Proceedings of the VLDB Endowment* 11.11 (2018), pages 1454–1467.

[Eic+17] Philipp Eichmann, Andrew Crotty, Alexander Galakatos, and Emanuel Zgraggen. "Discrete Time Specifications In Temporal Queries". In: *CHI Extended Abstracts*. 2017, pages 2536–2542.

[ElS+18] Hady ElSahar, Pavlos Vougiouklis, Arslen Remaci, Christophe Gravier, Jonathon S. Hare, Frédérique Laforest, and Elena Simperl. "T-REx: A Large Scale Alignment of Natural Language with Knowledge Base Triples". In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018*. European Language Resources Association (ELRA), 2018. URL: http://www.lrec-conf.org/proceedings/lrec2018/summaries/632.html.

[ER04] Günes Erkan and Dragomir R Radev. "Lexrank: Graph-based lexical centrality as salience in text summarization". In: *Journal of artificial intelligence research* 22 (2004), pages 457–479.

[FK14] James H. Faghmous and Vipin Kumar. "A Big Data Guide to Understanding Climate Change: The Case for Theory-Guided Data Science". In: *Big Data* 2.3 (Sept. 2014), pages 155–163. ISSN: 2167-6461. DOI: 10.1089/big.2014.0026. (Visited on 10/25/2023).

[FG17] Tobias Falke and Iryna Gurevych. "GraphDocExplore: A Framework for the Experimental Comparison of Graph-based Document Exploration Techniques". In: *EMNLP*. Copenhagen, Denmark: ACL, 2017, pages 19–24.

[Far+16] Manaal Faruqui, Yulia Tsvetkov, Pushpendre Rastogi, and Chris Dyer. "Problems with evaluation of word embeddings using word similarity tasks". In: *arXiv preprint arXiv:1605.02276* (2016).

[Fer+18] Raul Castro Fernandez, Essam Mansour, Abdulhakim A Qahtan, Ahmed Elmagarmid, Ihab Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. "Seeping semantics: Linking datasets using word embeddings for data discovery". In: *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE. 2018, pages 989–1000.

[Fin+18] Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir R. Radev. "Improving Text-to-SQL Evaluation Methodology". In: *ACL*. Volume abs/1806.09029. 2018, pages 351–360.

[Fri+17] Jason A. Fries, Sen Wu, Alexander Ratner, and Christopher Ré. "SwellShark: A Generative Model for Biomedical Named Entity Recognition without Labeled Data". In: *CoRR* abs/1704.06360 (2017). arXiv: 1704.06360.

[Ful82] R. Buckminster Fuller. *Critical Path*. St. Martin's Publishing Group, Feb. 15, 1982. 514 pages. ISBN: 978-0-312-17491-0.

[Gal+17] Alex Galakatos, Andrew Crotty, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. "Revisiting Reuse for Approximate Query Processing". In: *PVLDB* 10.10 (2017), pages 1142–1153.

[Gao+23] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. "Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation". In: (Sept. 20, 2023). DOI: 10.48550/arXiv.2308.15363. arXiv: 2308.15363 [cs]. preprint.

[GMG18] Yang Gao, Christian M. Meyer, and Iryna Gurevych. “APRIL: Interactively Learning to Summarise by Combining Active Preference Learning and Reinforcement Learning”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium, 2018, pages 4120–4130. DOI: 10.18653/v1/D18-1445. URL: https://www.aclweb.org/anthology/D18-1445.

[Gas+22] Marius Gassen, Benjamin Hättasch, Benjamin Hilprecht, Nadja Geisler, Alexander Fraser, and Carsten Binnig. “Demonstrating CAT: Synthesizing Data-Aware Conversational Agents for Transactional Databases”. In: *Proc. VLDB Endow.* 15.12 (2022), pages 3586–3589. URL: https://www.vldb.org/pvldb/vol15/p3586-h%5C%e4ttasch.pdf.

[GDR18] Sebastian Gehrmann, Yuntian Deng, and Alexander Rush. “Bottom-Up Abstractive Summarization”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Brussels, Belgium, 2018, pages 4098–4109. URL: https://www.aclweb.org/anthology/D18-1443.

[Gen+17] Anna Lisa Gentile, Petar Ristoski, Steffen Eckel, Dominique Ritze, and Heiko Paulheim. “Entity Matching on Web Tables: a Table Embeddings approach for Blocking.” In: *EDBT*. 2017, pages 510–513.

[Geo05] David George. “Understanding structural and semantic heterogeneity in the context of database schema integration”. In: *Journal of the Department of Computing, UCLAN* 4 (2005), pages 29–44.

[GB22] Alireza Ghadimi and Hamid Beigy. “Hybrid Multi-Document Summarization Using Pre-Trained Language Models”. In: *Expert Systems with Applications* 192 (Apr. 15, 2022), page 116292. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2021.116292. URL: https://www.sciencedirect.com/science/article/pii/S0957417421015979 (visited on 10/28/2023).

[GF09] Dan Gillick and Benoit Favre. “A scalable global model for summarization”. In: *Proceedings of the Workshop on Integer Linear Programming for Natural Langauge Processing*. Association for Computational Linguistics. 2009, pages 10–18.

[Gki+21] Orest Gkini, Theofilos Belmpas, Georgia Koutrika, and Yannis Ioannidis. “An In-Depth Benchmarking of Text-to-SQL Systems”. In: *Proceedings of the 2021 International Conference on Management of Data*. SIGMOD/PODS ’21. Virtual Event, China: Association for Computing Machinery, 2021, pages 632–644. ISBN: 9781450383431. DOI: 10.1145/3448016.3452836.

[Glo+13] Dorota Glowacka, Tuukka Ruotsalo, Ksenia Konuyshkova, Kumaripaba Athukorala, Kaski Samuel, and Giulio Jacucci. “Directing Exploratory Search: Reinforcement Learning from User Interactions with Keywords”. In: *ACM IUI*. Santa Monica, CA, USA: ACM, 2013, pages 117–127. URL: https://www.cs.helsinki.fi/u/jacucci/directing.pdf.

[GS96] Ralph Grishman and Beth Sundheim. “Message Understanding Conference - 6: A Brief History”. In: *16th International Conference on Computational Linguistics, Proceedings of the Conference, COLING 1996, Center for Sprogteknologi, Copenhagen, Denmark, August 5-9, 1996*. 1996, pages 466–471. URL: https://aclanthology.org/C96-1079/.

[GD18] Dagmar Gromann and Thierry Declerck. “Comparing pretrained multilingual word embeddings on an ontology alignment task”. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. 2018.

[Guo+19] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. “Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation”. In: *Proceeding of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*. Association for Computational Linguistics. 2019, pages 4524–4535.

[GB19] Dhruv Gupta and Klaus Berberich. “JIGSAW: Structuring Text into Tables”. In: *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*. ICTIR ’19. New York, NY, USA: Association for Computing Machinery, Sept. 2019, pages 237–244. ISBN: 978-1-4503-6881-0. DOI: 10.1145/3341981.3344228. (Visited on 10/20/2023).

[HV09] Aria Haghighi and Lucy Vanderwende. “Exploring content models for multi-document summarization”. In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics. 2009, pages 362–370.

[Han+17] Maeda F. Hanafi, Azza Abouzied, Laura Chiticariu, and Yunyao Li. “SEER: Auto-Generating Information Extraction Rules from User-Specified Examples”. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI ’17. Denver, Colorado, USA: Association for Computing Machinery, 2017, pages 6672–6682. ISBN: 9781450346559. DOI: 10.1145/3025453.3025540.

[Har+20] L. Harper, N. Kalfa, G.M.A. Beckers, M. Kaefer, A.J. Nieuwhof-Leppink, Magdalena Fossum, K.W. Herbst, and D. Bagli. “The Impact of COVID-19 on Research”. In: *Journal of Pediatric Urology* 16.5 (Oct. 2020), pages 715–716. ISSN: 1477-5131. DOI: 10.1016/j.jpurol.2020.07.002. pmid: 32713792. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7343645/ (visited on 10/27/2023).

[Har54] Zellig S Harris. “Distributional structure”. In: *Word-journal of The International Linguistic Association* 10.2-3 (1954), pages 146–162.

[Hät18] Benjamin Hättasch. “Towards Interactive Summarization of Large Document Collections”. In: *Proceedings of the First Biennial Conference on Design of Experimental Search & Information Retrieval Systems, Bertinoro, Italy, August 28-31, 2018*. Edited by Omar Alonso and Gianmaria Silvello. Volume 2167. CEUR Workshop Proceedings. CEUR-WS.org, 2018, page 103. URL: https://ceur-ws.org/Vol-2167/short6.pdf.

[Hät21] Benjamin Hättasch. “WannaDB: Ad-hoc Structured Exploration of Text Collections Using Queries”. In: *Proceedings of the Second International Conference on Design of Experimental Search & Information REtrieval Systems, Padova, Italy, September 15-18, 2021*. Edited by Omar Alonso, Stefano Marchesin, Marc Najork, and Gianmaria Silvello. Volume 2950. CEUR Workshop Proceedings. CEUR-WS.org, 2021, pages 179–180. URL: https://ceur-ws.org/Vol-2950/paper-23.pdf.

[HBB21] Benjamin Hättasch, Jan-Micha Bodensohn, and Carsten Binnig. “ASET: Ad-hoc Structured Exploration of Text Collections”. en. In: *3rd International Workshop on Applied AI for Database Systems and Applications (AIDB21). In conjunction with the 47th International Conference on Very Large Data Bases, Copenhagen, Denmark, August 16 - 20, 2021*. Copenhagen, Denmark, 2021. arXiv: 2203.04663.

[HBB22] Benjamin Hättasch, Jan-Micha Bodensohn, and Carsten Binnig. “Demonstrating ASET: Ad-hoc Structured Exploration of Text Collections”. In: *SIGMOD ’22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. ACM, 2022, pages 2393–2396. DOI: 10.1145/3514221.3520174.

[Hät+23a] Benjamin Hättasch, Jan-Micha Bodensohn, Liane Vogel, Matthias Urban, and Carsten Binnig. “WannaDB: Ad-hoc SQL Queries over Text Collections”. In: *Datenbanksysteme für Business, Technologie und Web (BTW 2023), 20. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme” (DBIS), 06.-10, März 2023, Dresden, Germany, Proceedings*. Edited by Birgitta König-Ries, Stefanie Scherzinger, Wolfgang Lehner, and Gottfried Vossen. Volume P-331. LNI. Gesellschaft für Informatik e.V., 2023, pages 157–181. DOI: 10.18420/BTW2023-08.

[HGB21] Benjamin Hättasch, Nadja Geisler, and Carsten Binnig. “Netted?! How to Improve the Usefulness of Spider & Co”. In: *Proceedings of the Second International Conference on Design of Experimental Search & Information REtrieval Systems, Padova, Italy, September 15-18, 2021*. Edited by Omar Alonso, Stefano Marchesin, Marc Najork, and Gianmaria Silvello. Volume 2950. CEUR Workshop Proceedings. CEUR-WS.org, 2021, pages 38–43. URL: https://ceur-ws.org/Vol-2950/paper-08.pdf.

[Hät+20a] Benjamin Hättasch, Nadja Geisler, Christian M. Meyer, and Carsten Binnig. “Summarization Beyond News: The Automatically Acquired Fandom Corpora”. In: *Proceedings of The 12th Language Resources and Evaluation Conference, LREC 2020, Marseille, France, May 11-16, 2020*. Edited by Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asunción Moreno, Jan Odijk, and Stelios Piperidis. European Language Resources Association, 2020, pages 6700–6708. URL: https://aclanthology.org/2020.lrec-1.827/.

[HMB19] Benjamin Hättasch, Christian M. Meyer, and Carsten Binnig. “Interactive Summarization of Large Document Collections”. In: *Proceedings of the Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD 2019, Amsterdam, The Netherlands, July 5, 2019*. ACM, 2019, 9:1–9:4. DOI: 10.1145/3328519.3329129.

[Hät+20b] Benjamin Hättasch, Michael Truong-Ngoc, Andreas Schmidt, and Carsten Binnig. “It’s AI Match: A Two-Step Approach for Schema Matching Using Embeddings”. In: *2nd International Workshop on Applied AI for Database Systems and Applications (AIDB20). In conjunction with the 46th International Conference on Very Large Data Bases, Virtual, August 31 - September 4, 2020*. Edited by Bingsheng He, Berthold Reinwald, and Yingjun Wu. Virtual, 2020. DOI: 10.48550/arXiv.2203.04366.

[Hät+23b] Benjamin Hättasch, Liane Vogel, Gard Jenset, Jan-Micha Bodensohn, Chandrima Roy, and Carsten Binnig. “WannaDB in Action: Deploying Ad-hoc SQL-over-Text Exploration in an Industrial Scenario”. In: *Under submission* (2023).

[HGD90] Charles T. Hemphill, John J. Godfrey, and George R. Doddington. “The ATIS Spoken Language Systems Pilot Corpus”. In: *Proceedings of the Workshop on Speech and Natural Language*. HLT ’90. Hidden Valley, Pennsylvania: Association for Computational Linguistics, 1990, pages 96–101. DOI: 10.3115/116580.116613.

[Hen+21] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. “Measuring Mathematical Problem Solving With the MATH Dataset”. In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*. 2021. URL: https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

[Her+15] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. “Teaching machines to read and comprehend”. In: *Advances in Neural Information Processing Systems (NIPS)*. Montréal, Canada, 2015, pages 1684–1692. URL: http://papers.nips.cc/paper/5945-teaching-machines-to-read-and-comprehend.pdf.

[Her+20] Daniel Ayala Hernández, Inma Hernández, David Ruiz, and Erhard Rahm. “LEAPME: Learning-based Property Matching with Embeddings”. In: *CoRR* abs/2010.01951 (2020). arXiv: 2010.01951.

[Het23] Ewald Hetrodt. *Mit Künstlicher Intelligenz regieren*. F.A.Z. Aug. 16, 2023. URL: https://zeitung.faz.net/faz/rhein-main/2023-08-16/mit-kuenstlicher-intelligenz-regieren/927101.html (visited on 10/27/2023).

[HRK15] Felix Hill, Roi Reichart, and Anna Korhonen. “Simlex-999: Evaluating semantic models with (genuine) similarity estimation”. In: *Computational Linguistics* 41.4 (2015), pages 665–695.

[Hon+14] Kai Hong, John M Conroy, Benoit Favre, Alex Kulesza, Hui Lin, and Ani Nenkova. “A Repository of State of the Art and Competitive Baseline Summaries for Generic News Summarization.” In: *LREC*. 2014, pages 1608–1616.

[Hon+20] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. *spaCy: Industrial-strength Natural Language Processing in Python*. 2020. DOI: 10.5281/zenodo.1212303. URL: https://github.com/explosion/spaCy.

[HAE16] Mi-Young Huh, Pulkit Agrawal, and Alexei A. Efros. “What makes ImageNet good for transfer learning?” In: *CoRR* abs/1608.08614 (2016).

[Hul97] Richard Hull. “Managing semantic heterogeneity in databases: a theoretical prospective”. In: *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. 1997, pages 51–61.

[IC19] Ihab F. Ilyas and Xu Chu. *Data Cleaning*. Morgan & Claypool, June 18, 2019. 284 pages. ISBN: 978-1-4503-7155-1. Google Books: RxieDwAAQBAJ.

[II08] Aminul Islam and Diana Inkpen. “Semantic text similarity using corpus-based word similarity and string similarity”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 2.2 (2008), pages 1–25.

[Iye+17] Srinivasan Iyer et al. “Learning a Neural Semantic Parser from User Feedback”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*. 2017, pages 963–973.

[Jaw12] Kalpana V Jawale. “Methods of sampling design in the legal research: Advantages and disadvantages”. In: *Online International Interdisciplinary Research Journal* 2.6 (2012), pages 183–190.

[JL16] Robin Jia and Percy Liang. “Data Recombination for Neural Semantic Parsing”. In: *ACL*. Volume abs/1606.03622. 2016.

[JPP17] Rogers Jeffrey Leo John, Navneet Potti, and Jignesh M Patel. “Ava: From Data to Insights Through Conversations.” In: *CIDR*. 2017.

[KL10] Pamela Karr-Wisniewski and Ying Lu. “When more is too much: Operationalizing technology overload and exploring its impact on knowledge worker productivity”. In: *Computers in Human Behavior* 26.5 (2010). Advancing Educational Research on Computer-supported Collaborative Learning (CSCL) through the use of gStudy CSCL Tools, pages 1061–1072. ISSN: 0747-5632. DOI: 10.1016/j.chb.2010.03.008. URL: https://www.sciencedirect.com/science/article/pii/S0747563210000488.

[KK23] George Katsogiannis-Meimarakis and Georgia Koutrika. “A Survey on Deep Learning Approaches for Text-to-SQL”. In: *The VLDB Journal* 32.4 (July 1, 2023), pages 905–936. ISSN: 0949-877X. DOI: 10.1007/s00778-022-00776-8. (Visited on 10/23/2023).

[KMD18] Chris Kedzie, Kathleen McKeown, and Hal Daumé III. “Content Selection in Deep Learning Models of Summarization”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pages 1818–1828. DOI: 10.18653/v1/D18-1208. URL: https://www.aclweb.org/anthology/D18-1208.

[KR15] Tom Kenter and Maarten de Rijke. “Short Text Similarity with Word Embeddings”. In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. CIKM ’15. Melbourne, Australia: ACM, 2015, pages 1411–1420. ISBN: 978-1-4503-3794-6. DOI: 10.1145/2806416.2806475.

[KCP18] Kian Kenyon-Dean, Jackie Chi Kit Cheung, and Doina Precup. “Resolving Event Coreference with Supervised Representation Learning and Clustering-Oriented Regularization”. In: *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics, *SEM@NAACL-HLT 2018, New Orleans, Louisiana, USA, June 5-6, 2018*. Association for Computational Linguistics, 2018, pages 1–10. DOI: 10.18653/v1/s18-2001.

[Kho+17] Mahnoosh Kholghi, Lance De Vine, Laurianne Sitbon, Guido Zuccon, and Anthony N. Nguyen. “Clinical information extraction using small data: An active learning approach based on sequence representations and word embeddings”. In: *J. Assoc. Inf. Sci. Technol.* 68.11 (2017), pages 2543–2556. DOI: 10.1002/asi.23936.

[Kim+20] Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. “Natural Language to SQL: Where Are We Today?” en. In: *Proceedings of the VLDB Endowment* 13.10 (2020), pages 1737–1750. ISSN: 2150-8097. DOI: 10.14778/3401960.3401970.

[KKK18] Prodromos Kolyvakis, Alexandros Kalousis, and Dimitris Kiritsis. “Deepalignment: Unsupervised ontology matching with refined word vectors”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. 2018, pages 787–798.

[Kov+22] Ádám Kovács, Kinga Gémes, Eszter Iklódi, and Gábor Recski. “POTATO: exPlainable infOrmation exTrAcTion framewOrk”. In: *CoRR* abs/2201.13230 (2022). arXiv: 2201.13230.

[LAW21] Séamus Lankford, Haithem Afli, and Andy Way. “Transformers for Low-Resource Languages: Is Féidir Linn!” In: Machine Translation Summit XVIII: Research Track. Virtual: Association for Machine Translation in the Americas (AMTA), Aug. 16, 2021. URL: https://aclanthology.org/2021.mtsummit-research.5 (visited on 10/30/2023).

[LM14] Quoc Le and Tomas Mikolov. “Distributed representations of sentences and documents”. In: *International conference on machine learning*. 2014, pages 1188–1196.

[Lee+09] Chiw Yi Lee, Hamidah Ibrahim, Mohamed Othman, and Razali Yaakob. “Reconciling semantic conflicts in electronic patient data exchange”. In: *Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services*. 2009, pages 390–394.

[Lem+20] Domenico Lembo, Yunyao Li, Lucian Popa, and Federico Maria Scafoglieri. “Ontology mediated information extraction in financial domain with Mastro System-T”. In: *Proceedings of the Sixth International Workshop on Data Science for Macro-Modeling, DSMM 2020, In conjunction with the ACM SIGMOD/PODS Conference, Portland, OR, USA, June 14, 2020*. ACM, 2020, 3:1–3:6. DOI: 10.1145/3401832.3402681.

[LN07] Ulf Leser and Felix Naumann. *Informationsintegration - Architekturen und Methoden zur Integration verteilter heterogener Datenquellen*. dpunkt.verlag, 2007.

[Lew+20] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. “BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*. Association for Computational Linguistics, 2020, pages 7871–7880. DOI: 10.18653/v1/2020.acl-main.703.

[LJ14a] Fei Li and H. V. Jagadish. “Constructing an Interactive Natural Language Interface for Relational Databases”. In: *PVLDB* 8 (2014), pages 73–84. DOI: 10.14778/2735461.2735468.

[LJ14b] Fei Li and Hosagrahar Visvesvaraya Jagadish. “NaLIR: an interactive natural language interface for querying relational databases”. In: *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*. 2014, pages 709–712.

[Li+23] Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. *Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs*. May 30, 2023. DOI: 10.48550/arXiv.2305.03111. arXiv: 2305.03111 [cs]. preprint.

[Lia+20] Chen Liang, Yue Yu, Haoming Jiang, Siawpeng Er, Ruijia Wang, Tuo Zhao, and Chao Zhang. “BOND: BERT-Assisted Open-Domain Named Entity Recognition with Distant Supervision”. In: *KDD ’20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*. ACM, 2020, pages 1054–1064. DOI: 10.1145/3394486.3403149.

[LJK11] Percy Liang, Michael I. Jordan, and Dan Klein. “Learning Dependency-Based Compositional Semantics”. In: *ACL*. 2011, pages 590–599.

[LSX20] Xi Victoria Lin, Richard Socher, and Caiming Xiong. *Bridging Textual and Tabular Data for Cross-Domain Text-to-SQL Semantic Parsing*. 2020. arXiv: 2012.12627 [cs.CL].

[Liu+23] Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S. Yu. *A Comprehensive Evaluation of ChatGPT’s Zero-Shot Text-to-SQL Capability*. Mar. 11, 2023. DOI: 10.48550/arXiv.2303.13547. arXiv: 2303.13547 [cs]. preprint.

[LH14] Zhicheng Liu and Jeffrey Heer. “The Effects of Interactive Latency on Exploratory Visual Analysis”. In: *IEEE transactions on visualization and computer graphics* 20.12 (2014), pages 2122–2131. DOI: 10.1109/TVCG.2014.2346452.

[Luh58] Hans Peter Luhn. “The automatic creation of literature abstracts”. In: *IBM Journal of research and development* 2.2 (1958), pages 159–165.

[Lyo+16] Gabriel Lyons, Vinh Tran, Carsten Binnig, Ugur Çetintemel, and Tim Kraska. “Making the Case for Query-by-Voice with EchoQuery”. In: *SIGMOD*. 2016, pages 2129–2132.

[Ma+07] Linxiao Ma, John Ferguson, Marc Roper, and Murray Wood. “Investigating the Viability of Mental Models Held by Novice Programmers”. In: *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. SIGCSE ’07. New York, NY, USA: Association for Computing Machinery, Mar. 7, 2007, pages 499–503. ISBN: 978-1-59593-361-4. DOI: 10.1145/1227310.1227481. (Visited on 10/27/2023).

[MT14] Ahmed Mounaf Mahdi and Sabrina Tiun. “Utilizing wordnet and regular expressions for instance-based schema matching”. In: *Research Journal of Applied Sciences, Engineering and Technology* 8.4 (2014), pages 460–470.

[Man+14] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. “The Stanford CoreNLP Natural Language Processing Toolkit”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, System Demonstrations*. The Association for Computer Linguistics, 2014, pages 55–60. DOI: 10.3115/v1/p14-5010.

[May+20] Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan T. McDonald. “On Faithfulness and Factuality in Abstractive Summarization”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*. Association for Computational Linguistics, 2020, pages 1906–1919. DOI: 10.18653/v1/2020.acl-main.173.

[MIA17] Osama A Mehdi, Hamidah Ibrahim, and Lilly Suriani Affendey. “An approach for instance based schema matching with google similarity and regular expression.” In: *Int. Arab J. Inf. Technol.* 14.5 (2017), pages 755–763.

[Mik+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. “Distributed Representations of Words and Phrases and their Compositionality”. In: (2013). arXiv: 1310.4546 [cs.CL].

[Mik+18] Tomás Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. “Advances in Pre-Training Distributed Word Representations”. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018*. European Language Resources Association (ELRA), 2018. URL: http://www.lrec-conf.org/proceedings/lrec2018/summaries/721.html.

[Mor+19] Mário W. L. Moreira, Joel J. P. C. Rodrigues, Valery Korotaev, Jalal Al-Muhtadi, and Neeraj Kumar. “A Comprehensive Review on Smart Decision Support Systems for Health Care”. In: *IEEE Systems Journal* 13.3 (Sept. 2019), pages 3536–3545. ISSN: 1937-9234. DOI: 10.1109/JSYST.2018.2890121. URL: https://ieeexplore.ieee.org/abstract/document/8611153 (visited on 10/26/2023).

[Mug02] Fridah W Mugo. “Sampling in research”. In: (2002).

[NZZ17] Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. “SummaRuNNer: A Recurrent Neural Network Based Sequence Model for Extractive Summarization of Documents”. In: *Proceedings of the Thirty-First Conference on Artificial Intelligence (AAAI)*. San Francisco, CA, USA, 2017, pages 3075–3081. URL: https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14636/.

[NGV12] Courtney Napoles, Matthew Gormley, and Benjamin Van Durme. “Annotated Gigaword”. In: *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction (AKBC/WEKEX)*. Montréal, Canada, 2012, pages 95–100. URL: https://www.aclweb.org/anthology/W12-3018.

[Nar+22] Avanika Narayan, Ines Chami, Laurel Orr, Simran Arora, and Christopher Ré. *Can Foundation Models Wrangle Your Data?* Dec. 24, 2022. DOI: 10.48550/arXiv.2205.09911. arXiv: 2205.09911 [cs]. preprint.

[NCL18] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. “Ranking Sentences for Extractive Summarization with Reinforcement Learning”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL/HLT): Long Papers*. New Orleans, LA, USA, 2018, pages 1747–1759. DOI: 10.18653/v1/N18-1158.

[NM11] Ani Nenkova and Kathleen McKeown. “Automatic Summarization”. In: *Foundations and Trends® in Information Retrieval* 5.2–3 (2011), pages 103–233. ISSN: 1554-0669. DOI: 10.1561/1500000015.

[Ngu+17] Dat Ba Nguyen, Abdalghani Abujabal, Khanh Tran, Martin Theobald, and Gerhard Weikum. “Query-Driven On-The-Fly Knowledge Base Construction”. In: *Proc. VLDB Endow.* 11.1 (2017), pages 66–79. DOI: 10.14778/3151113.3151119. URL: http://www.vldb.org/pvldb/vol11/p66-nguyen.pdf.

[Ngu+20] Minh-Tien Nguyen, Dung Tien Le, Nguyen Hong Son, Bui Cong Minh, Do Hoang Thai Duong, and Le Thai Linh. “Understanding Transformers for Information Extraction with Limited Data”. In: *Proceedings of the 34th Pacific Asia Conference on Language, Information and Computation, PACLIC 2020, Hanoi, Vietnam, October 24-26, 2020*. Association for Computational Linguistics, 2020, pages 478–487. URL: https://aclanthology.org/2020.paclic-1.55/.

[NHN19] Kenji Nozaki, Teruhisa Hochin, and Hiroki Nomiya. “Semantic Schema Matching for String Attribute with Word Vectors and its Evaluation”. In: *International Journal of Networked and Distributed Computing* (Jan. 2019). DOI: 10.2991/ijndc.k.190710.001.

[Owc+12] Karolina Owczarzak, John M Conroy, Hoa Trang Dang, and Ani Nenkova. “An assessment of the accuracy of automatic evaluation in summarization”. In: *Proceedings of Workshop on Evaluation Metrics and System Comparison for Automatic Summarization*. Association for Computational Linguistics. 2012, pages 1–9.

[Par+19] Yongjoo Park, Jingyi Qing, Xiaoyang Shen, and Barzan Mozafari. “BlinkML: Approximate Machine Learning with Probabilistic Guarantees”. In: *SIGMOD*. (to appear). 2019.

[PKT09] Jeffrey Partyka, Latifur Khan, and Bhavani Thuraisingham. “Semantic schema matching without shared instances”. In: *2009 IEEE International Conference on Semantic Computing*. IEEE. 2009, pages 297–302.

[PXS18] Romain Paulus, Caiming Xiong, and Richard Socher. “A Deep Reinforced Model for Abstractive Summarization”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. Vancouver, Canada, 2018. URL: https://openreview.net/forum?id=HkAClQgA-.

[PC16] Ellie Pavlick and Chris Callison-Burch. “Simple PPDB: A Paraphrase Database for Simplification”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*. 2016.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. “Glove: Global Vectors for Word Representation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2014, pages 1532–1543. DOI: 10.3115/v1/d14-1162.

[Pet+18] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. “Deep contextualized word representations”. In: *arXiv preprint arXiv:1802.05365* (2018).

[PE16] Maxime Peyrard and Judith Eckle-Kohler. “Optimizing an Approximation of ROUGE - a Problem-Reduction Approach to Extractive Multi-Document Summarization”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pages 1825–1836. DOI: 10.18653/v1/P16-1172. URL: https://www.aclweb.org/anthology/P16-1172.

[Pin+15] Christoph Pinkel, Carsten Binnig, Ernesto Jiménez-Ruiz, Wolfgang May, Dominique Ritze, Martin G. Skjæveland, Alessandro Solimando, and Evgeny Kharlamov. “RODI: A Benchmark for Automatic Mapping Generation in Relational-to-Ontology Data Integration”. In: *The Semantic Web. Latest Advances and New Domains - 12th European Semantic Web Conference, ESWC 2015, Portoroz, Slovenia, May 31 - June 4, 2015. Proceedings*. 2015, pages 21–37.

[Piw+18] Heather Piwowar, Jason Priem, Vincent Larivière, Juan Pablo Alperin, Lisa Matthias, Bree Norlander, Ashley Farley, Jevin West, and Stefanie Haustein. “The State of OA: A Large-Scale Analysis of the Prevalence and Impact of Open Access Articles”. In: *PeerJ* 6 (Feb. 13, 2018), e4375. ISSN: 2167-8359. DOI: 10.7717/peerj.4375. pmid: 29456894. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5815332/ (visited on 10/26/2023).

[Pop+03] Ana-Maria Popescu et al. “Towards a theory of natural language interfaces to databases”. In: *IUI*. 2003.

[Pop+04] Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. “Modern Natural Language Interfaces to Databases: Composing Statistical Parsing with Semantic Tractability”. In: *COLING*. 2004.

[PEK03] Ana-Maria Popescu, Oren Etzioni, and Henry A. Kautz. “Towards a theory of natural language interfaces to databases”. In: *IUI*. 2003, pages 149–157.

[Pre16] Gil Press. *Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says*. Forbes. Mar. 23, 2016. URL: https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/ (visited on 10/28/2023).

[Qi+20] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. “Stanza: A Python Natural Language Processing Toolkit for Many Human Languages”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, ACL 2020, Online, July 5-10, 2020*. Association for Computational Linguistics, 2020, pages 101–108. DOI: 10.18653/v1/2020.acl-demos.14.

[Qin+22] Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, Fei Huang, and Yongbin Li. *A Survey on Text-to-SQL Parsing: Concepts, Methods, and Future Directions*. Aug. 29, 2022. DOI: 10.48550/arXiv.2208.13629. arXiv: 2208.13629 [cs]. preprint.

[RB01] Erhard Rahm and Philip A Bernstein. "A survey of approaches to automatic schema matching". In: *the VLDB Journal* 10.4 (2001), pages 334–350.

[RJL18] Pranav Rajpurkar, Robin Jia, and Percy Liang. "Know What You Don't Know: Unanswerable Questions for SQuAD". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*. Association for Computational Linguistics, 2018, pages 784–789. DOI: 10.18653/v1/P18-2124. URL: https://aclanthology.org/P18-2124/.

[Ran+05a] Rodolfo A. Pazos Rangel, O. Joaquín Pérez, B. Juan Javier González, Alexander Gelbukh, Grigori Sidorov, and M. Myriam J. Rodríguez. "A Domain Independent Natural Language Interface to Databases Capable of Processing Complex Queries". In: *MICAI 2005: Advances in Artificial Intelligence*. Edited by Alexander Gelbukh, Álvaro de Albornoz, and Hugo Terashima-Marín. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pages 833–842. ISBN: 978-3-540-31653-4.

[Ran+05b] Rodolfo A. Pazos Rangel, Joaquín Pérez Ortega, Juan Javier González Barbosa, Alexander F. Gelbukh, Grigori Sidorov, and Myriam J. Rodríguez M. "A Domain Independent Natural Language Interface to Databases Capable of Processing Complex Queries". In: *MICAI*. Volume 3789. 2005, pages 833–842.

[Rat+17a] Alexander Ratner, Stephen H. Bach, Henry R. Ehrenberg, Jason Alan Fries, Sen Wu, and Christopher Ré. "Snorkel: Rapid Training Data Creation with Weak Supervision". In: *PVLDB* 11.3 (2017), pages 269–282.

[Rat+17b] Alexander J. Ratner, Henry R. Ehrenberg, Zeshan Hussain, Jared Dunnmon, and Christopher Ré. "Learning to Compose Domain-Specific Transformations for Data Augmentation". In: *NIPS*. 2017, pages 3236–3246.

[Rec+21] Gábor Recski, Björn Lellmann, Ádám Kovács, and Allan Hanbury. "Explainable Rule Extraction via Semantic Graphs". In: *Joint Proceedings of the Workshops on Automated Semantic Analysis of Information in Legal Text (ASAIL 2021) and AI and Intelligent Assistance for Legal Professionals in the Digital Workplace (LegalAIIA 2021) held online in conjunction with 18th International Conference on Artificial Intelligence and Law (ICAIL 2021)*. Volume 2888. CEUR Workshop Proceedings. Sao Paolo, Brazil (held online): CEUR-WS.org, 2021, pages 24–35. URL: http://ceur-ws.org/Vol-2888/paper3.pdf.

[RG19a] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Association for Computational Linguistics, 2019, pages 3980–3990. DOI: 10.18653/v1/D19-1410.

[RG19b] Nils Reimers and Iryna Gurevych. “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Nov. 2019, pages 3982–3992. arXiv: 1908.10084.

[RV13] Cristobal Romero and Sebastian Ventura. “Data Mining in Education”. In: *WIREs Data Mining and Knowledge Discovery* 3.1 (2013), pages 12–27. ISSN: 1942-4795. DOI: 10.1002/widm.1075. (Visited on 10/25/2023).

[Sa+16] Christopher De Sa, Alexander Ratner, Christopher Ré, Jaeho Shin, Feiran Wang, Sen Wu, and Ce Zhang. “DeepDive: Declarative Knowledge Base Construction”. In: *SIGMOD Rec.* 45.1 (2016), pages 60–67. DOI: 10.1145/2949741.2949756.

[Sah+16] Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R. Mittal, and Fatma Özcan. “ATHENA: An Ontology-Driven System for Natural Language Querying over Relational Data Stores”. In: *Proc. VLDB Endow.* 9.12 (2016), pages 1209–1220. ISSN: 2150-8097.

[SMJ19] Tanvi Sahay, Ankita Mehta, and Shruti Jadon. “Schema Matching using Machine Learning”. In: *arXiv preprint arXiv:1911.11543* (2019).

[Sai+22] Oscar Sainz, Haoling Qiu, Oier Lopez de Lacalle, Eneko Agirre, and Bonan Min. “ZS4IE: A toolkit for Zero-Shot Information Extraction with simple Verbalizations”. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: System Demonstrations*. Hybrid: Seattle, Washington + Online: Association for Computational Linguistics, July 2022, pages 27–38. URL: https://aclanthology.org/2022.naacl-demo.4.

[Sch13] David Russell Schilling. *Knowledge Doubling Every 12 Months, Soon to Be Every 12 Hours*. Industry Tap. Apr. 19, 2013. URL: https://www.industrytap.com/knowledge-doubling-every-12-months-soon-to-be-every-12-hours/3950 (visited on 10/26/2023).

[Sch+22] Johannes Schleith, Hella Hoffmann, Milda Norkute, and Brian Cechmanek. “Human in the loop information extraction increases efficiency and trust”. In: *Mensch und Computer 2022 - Workshopband*. Edited by Karola Marky, Uwe Grünefeld, and Thomas Kosch. Bonn: Gesellschaft für Informatik e.V., 2022. DOI: 10.18420/muc2022-mci-ws12-249.

[SLM17] Abigail See, Peter J. Liu, and Christopher D. Manning. “Get To The Point: Summarization with Pointer-Generator Networks”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL): Long Papers*. Vancouver, Canada, 2017, pages 1073–1083. DOI: 10.18653/v1/P17-1099.

[Sha+18] Pararth Shah, Dilek Hakkani-Tür, Gökhan Tür, Abhinav Rastogi, Ankur Bapna, Neha Nayak, and Larry P. Heck. “Building a Conversational Agent Overnight with Dialogue Self-Play”. In: *CoRR* abs/1801.04871 (2018). arXiv: 1801.04871.

[Sha+19] Zeyuan Shang, Emanuel Zgraggen, Benedetto Buratti, Ferdinand Kossmann, Philipp Eichmann, Yeounoh Chung, Carsten Binnig, Eli Upfal, and Tim Kraska. “Democratizing Data Science through Interactive Curation of ML Pipelines”. In: *SIGMOD*. 2019, pages 1171–1188.

[SJ19] Shreyas Sharma and Ron Daniel Jr. “BioFLAIR: Pretrained Pooled Contextualized Embeddings for Biomedical Sequence Labeling Tasks”. In: *CoRR* abs/1908.05760 (2019). arXiv: 1908.05760.

[Shi+20] Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cicero Nogueira dos Santos, and Bing Xiang. *Learning Contextual Representations for Semantic Parsing with Generation-Augmented Pre-Training*. 2020. arXiv: 2012.10309 [cs.CL].

[Sko+19] P. O. Skobelev, E. V. Simonova, S. V. Smirnov, D. S. Budaev, G. Yu. Voshchuk, and A. L. Morokov. “Development of a Knowledge Base in the “Smart Farming” System for Agricultural Enterprise Management”. In: *Procedia Computer Science*. Proceedings of the 13th International Symposium “Intelligent Systems 2018” (INTELS’18), 22-24 October, 2018, St. Petersburg, Russia 150 (Jan. 1, 2019), pages 154–161. ISSN: 1877-0509. DOI: 10.1016/j.procs.2019.02.029. URL: https://www.sciencedirect.com/science/article/pii/S1877050919303734 (visited on 10/26/2023).

[Smi+22] Ellery Smith, Dimitris Papadopoulos, Martin Braschler, and Kurt Stockinger. “LILLIE: Information extraction and database integration using linguistics and learning-based algorithms”. In: *Inf. Syst.* 105 (2022), page 101938. DOI: 10.1016/j.is.2021.101938.

[Soc+12] Richard Socher, Brody Huval, Christopher D Manning, and Andrew Y Ng. “Semantic compositionality through recursive matrix-vector spaces”. In: *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*. Association for Computational Linguistics. 2012, pages 1201–1211.

[SGL08] John Stasko, Carsten Görg, and Zhicheng Liu. “Jigsaw: Supporting Investigative Analysis through Interactive Visualization”. In: *Information Visualization* 7.2 (Apr. 2008), pages 118–132. ISSN: 1473-8716. DOI: 10.1145/1466620.1466622. (Visited on 10/20/2023).

[SJ04] Josef Steinberger and Karel Jezek. “Using latent semantic analysis in text summarization and summary evaluation”. In: *Proc. ISIM* 4 (2004), pages 93–100.

[Sun+17] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. “Revisiting Unreasonable Effectiveness of Data in Deep Learning Era”. In: *ICCV*. 2017, pages 843–852.

[Sun+23] Shuo Sun, Yuchen Zhang, Jiahuan Yan, Yuze Gao, Donovan Ong, Bin Chen, and Jian Su. *Battle of the Large Language Models: Dolly vs LLaMA vs Vicuna vs Guanaco vs Bard vs ChatGPT – A Text-to-SQL Parsing Comparison*. Oct. 16, 2023. DOI: 10.48550/arXiv.2310.10190. arXiv: 2310.10190 [cs]. preprint.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems*. NIPS’14. Montreal, Canada: MIT Press, 2014, pages 3104–3112. URL: http://dl.acm.org/citation.cfm?id=2969033.2969173.

[TAB] *Tableau*. http://www.tableau.com. Accessed: 2018-03-01.

[TL08] Sandeep Tata and Guy M. Lohman. “SQAK: doing more with keywords”. In: *SIGMOD*. 2008, pages 889–902.

[Ter+15] Pawel Terlecki et al. “On Improving User Response Times in Tableau”. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*. SIGMOD ’15. Melbourne, Victoria, Australia, 2015, pages 1695–1706.

[Tou+23] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. *LLaMA: Open and Efficient Foundation Language Models*. Feb. 27, 2023. DOI: 10.48550/arXiv.2302.13971. arXiv: 2302.13971 [cs]. preprint.

[Tur+18] Cagatay Turkay, Nicola Pezzotti, Carsten Binnig, Hendrik Strobelt, Barbara Hammer, Daniel A. Keim, Jean-Daniel Fekete, Themis Palpanas, Yunhai Wang, and Florin Rusu. “Progressive Data Science: Potential and Challenges”. In: *CoRR* abs/1812.08032 (2018).

[UR20] Matej Ulčar and Marko Robnik-Šikonja. *High Quality ELMo Embeddings for Seven Less-Resourced Languages*. Mar. 27, 2020. DOI: 10.48550/arXiv.1911.10049. arXiv: 1911.10049 [cs]. preprint.

[Uta+18] Prasetya Utama, Nathaniel Weir, Fuat Basik, Carsten Binnig, Ugur Cetintemel, Benjamin Hättasch, Amir Ilkhechi, Shekar Ramaswamy, and Arif Usta. *An End-to-end Neural Natural Language Interface for Databases*. 2018. arXiv: 1804.00401 [cs.DB].

[VMR11] Marta Vila, Maria Antònia Martí, and Horacio Rodríguez. “Paraphrase Concept and Typology. A Linguistically Based and Computationally Oriented Approach”. In: *Procesamiento del Lenguaje Natural* 46 (2011), pages 83–90.

[Vin+15] Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. “Grammar as a Foreign Language”. In: *NIPS*. 2015, pages 2773–2781.

[VHB23] Liane Vogel, Benjamin Hilprecht, and Carsten Binnig. *Towards Foundation Models for Relational Databases [Vision Paper]*. May 24, 2023. DOI: 10.48550/arXiv.2305.15321. arXiv: 2305.15321 [cs]. preprint.

[Wan+15] Yushi Wang et al. “Building a Semantic Parser Overnight”. In: *ACL*. 2015.

[WBL15] Yushi Wang, Jonathan Berant, and Percy Liang. “Building a Semantic Parser Overnight”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, July 2015, pages 1332–1342. DOI: 10.3115/v1/P15-1129. URL: https://www.aclweb.org/anthology/P15-1129.

[Wei+19a] Qiang Wei, Yukun Chen, Mandana Salimi, Joshua C. Denny, Qiaozhu Mei, Thomas A. Lasko, Qingxia Chen, Stephen Wu, Amy Franklin, Trevor Cohen, and Hua Xu. “Cost-aware active learning for named entity recognition in clinical text”. In: *J. Am. Medical Informatics Assoc.* 26.11 (2019), pages 1314–1322. DOI: 10.1093/jamia/ocz102.

[Wei+19b] Nathaniel Weir, Andrew Crotty, Alex Galakatos, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Ugur Cetintemel, Prasetya Utama, Nadja Geisler, Benjamin Hättasch, Steffen Eger, and Carsten Binnig. *DBPal: Weak Supervision for Learning a Natural Language Interface to Databases*. 2019. arXiv: 1909.06182 [cs.DB].

[Wei+20] Nathaniel Weir, Prasetya Utama, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Nadja Geisler, Benjamin Hättasch, Steffen Eger, et al. "DBPal: A Fully Pluggable NL2SQL Training Pipeline". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. Edited by David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo. ACM, 2020, pages 2347–2361. DOI: 10.1145/3318464.3380589.

[Wol+19] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. "HuggingFace's Transformers: State-of-the-art Natural Language Processing". In: *ArXiv* abs/1910.03771 (2019).

[Wor23] World Data Science Initiative. *Why Data Science Is the Most In-Demand Skill Now and How Can You Prepare for It?* 2023. URL: https://www.worlddatascience.org/blogs/why-data-science-is-the-most-indemand-skill-now-and-how-can-you-prepare-for-it (visited on 10/26/2023).

[WZL22] Xueqing Wu, Jiacheng Zhang, and Hang Li. "Text-to-Table: A New Way of Information Extraction". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*. Association for Computational Linguistics, 2022, pages 2518–2533. URL: https://aclanthology.org/2022.acl-long.180.

[XLS17] Xiaojun Xu, Chang Liu, and Dawn Song. "SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning". In: *CoRR* abs/1711.04436 (2017).

[YQC10] Jeffrey Xu Yu, Lu Qin, and Lijun Chang. "Keyword Search in Relational Databases: A Survey". In: *IEEE Data Eng. Bull.* 33.1 (2010), pages 67–78.

[Yu+18a] Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir R. Radev. "SyntaxSQLNet: Syntax Tree Networks for Complex and Cross-Domain Text-to-SQL Task". In: *EMNLP*. 2018, pages 1653–1663.

[Yu+19a] Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, and et al. "CoSQL: A Conversational Text-to-SQL Challenge Towards Cross-Domain Natural Language Interfaces to Databases". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (2019). DOI: 10.18653/v1/d19-1204.

[Yu+18b] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, 2018, pages 3911–3921.

[Yu+19b] Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Irene Li Heyang Er, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Vincent Zhang Jonathan Kraft, Caiming Xiong, Richard Socher, and Dragomir Radev. "SParC: Cross-Domain Semantic Parsing in Context". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, 2019, pages 4511–4523.

[ZM96] John M. Zelle and Raymond J. Mooney. "Learning to Parse Database Queries Using Inductive Logic Programming". In: *AAAI*. 1996, pages 1050–1055.

[ZC07] Luke S. Zettlemoyer and Michael Collins. "Online Learning of Relaxed CCG Grammars for Parsing to Logical Form". In: *EMNLP*. 2007, pages 678–687.

[Zgr+17] Emanuel Zgraggen, Alex Galakatos, Andrew Crotty, Jean-Daniel Fekete, and Tim Kraska. "How progressive visualizations affect exploratory analysis". In: *IEEE transactions on visualization and computer graphics* 23.8 (2017), pages 1977–1987.

[Zha+23] Jiani Zhang, Zhengyuan Shen, Balasubramaniam Srinivasan, Shen Wang, Huzefa Rangwala, and George Karypis. *NameGuess: Column Name Expansion for Tabular Data*. Oct. 19, 2023. DOI: 10.48550/arXiv.2310.13196. arXiv: 2310.13196 [cs]. preprint.

[Zha+24] Ming Zhang, Chengzhang Li, Meilin Wan, Xuejun Zhang, and Qingwei Zhao. "ROUGE-SEM: Better Evaluation of Summarization Using ROUGE Combined with Semantics". In: *Expert Systems with Applications* 237 (Mar. 1, 2024), page 121364. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2023.121364. URL: https://www.sciencedirect.com/science/article/pii/S0957417423018663 (visited on 10/29/2023).

[Zha+19] Rui Zhang, Tao Yu, He Yang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. "Editing-Based SQL Query Generation for Cross-Domain Context-Dependent Questions". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Hong Kong, China, 2019, pages 5338–5349.

[ZB18] Shuo Zhang and Krisztian Balog. "Ad Hoc Table Retrieval Using Semantic Similarity". In: *Proceedings of the 2018 World Wide Web Conference*. WWW '18. Lyon, France: International World Wide Web Conferences Steering Committee, 2018, pages 1553–1562. ISBN: 9781450356398. DOI: 10.1145/3178876.3186067.

[Zha+14] Yuanzhe Zhang, Xuepeng Wang, Siwei Lai, Shizhu He, Kang Liu, Jun Zhao, and Xueqiang Lv. "Ontology matching with word embeddings". In: *Chinese computational linguistics and natural language processing based on naturally annotated big data*. Springer, 2014, pages 34–45.

[ZE16] Tiancheng Zhao and Maxine Eskenazi. "Towards End-to-End Learning for Dialog State Tracking and Management using Deep Reinforcement Learning". In: *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. Los Angeles: Association for Computational Linguistics, Sept. 2016, pages 1–10. DOI: 10.18653/v1/W16-3601. URL: https://www.aclweb.org/anthology/W16-3601.

[ZXS17] Victor Zhong, Caiming Xiong, and Richard Socher. “Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning”. In: *CoRR* abs/1709.00103 (2017).

[ZC21] Zexuan Zhong and Danqi Chen. *A Frustratingly Easy Approach for Entity and Relation Extraction*. Mar. 23, 2021. DOI: 10.48550/arXiv.2010.12812. arXiv: 2010.12812 [cs]. preprint.

[Zop18] Markus Zopf. “Auto-hMDS: Automatic Construction of a Large Heterogeneous Multilingual Multi-Document Summarization Corpus”. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018*. 2018.