Accurate Performance Modeling for Distributed Stream Processing

Methods for Performance Benchmarking and Zero-shot Parallelism Tuning in Distributed and Heterogeneous Environments

at the Department of Electrical Engineering and Information Technology
of the Technische Universität Darmstadt

submitted in fulfillment of the requirements for the degree of
Doctor of Engineering (Dr.-Ing.)

Dissertation by Pratyush Agnihotri, M.Sc.,
born on 03.01.1988 in Kanpur, U.P., India

First examiner: Prof. Dr.-Ing. Ralf Steinmetz
Second examiner: Prof. Dr. Boris Koldehofe
Submitted on June 25, 2024
Disputation on September 13, 2024
Darmstadt 2024

Pratyush Agnihotri, M.Sc.: *Accurate Performance Modeling for Distributed Stream Processing,*
Methods for Performance Benchmarking and Zero-shot Parallelism Tuning in Distributed and Heterogeneous Environments

Darmstadt, Technische Universität Darmstadt

Jahr der Veröffentlichung der Dissertation auf TUprints: 2024
Year thesis published in TUprints: 2024

Tag der mündlichen Prüfung: September 13, 2024
Date of the oral examination: September 13, 2024

Bitte zitieren Sie dieses Dokument als: Please cite this document as:
URN: `urn:nbn:de:tuda-tuprints-281444`
URI: `https://tuprints.ulb.tu-darmstadt.de/id/eprint/28144`

"I dedicate this Ph.D. thesis to family who have always been the silent winds beneath my wings, guiding me invisibly through life's journey.
This accomplishment stands as a testament to your unwavering faith, endless sacrifice, and enduring love. You have instilled in me the courage to pursue knowledge, the resilience to persevere through challenges, and the humility to serve beyond myself.
This achievement is not just mine, but ours."

Abstract

Distributed Stream Processing (DSP) systems have emerged as a pivotal paradigm, enabling real-time data analysis using distributed cloud resources. Major Internet companies like Amazon and Google, build on DSP systems for their real-time data workloads. For instance, Amazon provides *Apache Flink* as a service for implementing DSP workloads. Parallelism is often a desired property of DSP workloads to meet the timeliness and scaling requirements of current applications, necessitating the use of distributed and multi-core cloud resources. However, cloud resources are *heterogeneous* in nature, which makes understanding the performance of DSP workloads very difficult, as it depends on highly varying resources, i.e., compute, storage, and network. Therefore, *(i)* understanding the performance and *(ii)* predicting it for distinct DSP workloads on such heterogeneous cloud environments are both very challenging problems. This thesis solves these two fundamental research challenges by contributing methods for accurate performance modeling of DSP workloads in heterogeneous cloud environments.

First, this thesis contributes to methods for performance understanding by proposing PDSP-Bench, a novel benchmarking system. It tackles three primary challenges of existing work: lack of expressiveness in benchmarking parallel DSP workloads, the need for heterogeneous hardware support, and the need for integration of learned DSP models. Unlike existing systems, PDSP-Bench enables the evaluation of parallel DSP applications and workloads using both synthetic and real-world applications, offering an expressive and scalable solution. Further, it facilitates the systematic training and evaluation of learned DSP models on diverse streaming workloads, which is crucial for optimizing performance. The extensive evaluation of PDSP-Bench demonstrates its benchmarking capabilities and highlights the impact of varying query complexities, hardware configurations, and workload parameters on system performance. The key observations of our experiments show the *non-linearity* and *paradoxical* effects of parallelism on performance.

Second, this thesis contributes to methods on performance prediction and optimization by proposing ZeroTune, a novel learned cost model for DSP workloads and an optimizer for parallelism tuning. It provides highly accurate cost predictions while generalizing to (unseen) heterogeneous hardware resources of the cloud. The generalizability of the model is based on *transfer learning*, the same technique that is used in *Large Language Models* like ChatGPT. The main idea is to learn from so-called *transferable features* and *parallel graph representation* that together enable the model to generalize to unseen DSP workloads and hardware. Our extensive evaluation demonstrates ZeroTune's robustness and accuracy across workloads, various parallelism degrees, unseen operator parameters, and training data efficiency. The evaluations show significant speed-ups with parallelism tuning compared to existing methods. Most notably, our approach has been adopted by *Amazon Redshift* for query execution time prediction.

Kurzfassung

Distributed-Stream-Processing (DSP) Systeme haben sich als zentrales Paradigma herauskristallisiert, das die Echtzeit-Datenanalyse mit verteilten Cloud-Ressourcen ermöglicht. Große Internetunternehmen wie Amazon und Google bauen auf DSP-Systeme für ihre Echtzeitdaten-Workloads. So bietet Amazon beispielsweise *Apache Flink* als Service für die Implementierung von DSP-Workloads an. Parallelität ist oft ein gewünschtes Merkmal von DSP-Workloads, um die Anforderungen an Aktualität und Skalierbarkeit heutiger Anwendungen zu erfüllen, was den Einsatz verteilter und Multi-Core-Cloud-Ressourcen erforderlich macht. Cloud-Ressourcen sind jedoch von Natur aus heterogen, was das Verständnis der Leistung von DSP-Workloads sehr erschwert, da sie von stark variierenden Ressourcen, d.h., Rechen-, Speicher- und Netzwerkressourcen, abhängen. Daher sind sowohl *(i)* das Verständnis der Leistung und *(ii)* die Vorhersage der Leistung für verschiedene DSP-Workloads in solchen heterogenen Cloud-Umgebungen sehr anspruchsvolle Probleme. Diese Arbeit löst diese beiden grundlegenden Forschungsherausforderungen, indem sie Methoden für eine genaue Leistungsmodellierung von DSP-Workloads in heterogenen Cloud-Umgebungen bereitstellt.

Erstens trägt diese Arbeit zu Methoden für das Verständnis der Leistung bei, indem sie PDSP-Bench vorschlägt, ein neuartiges Benchmarking-System. Es nimmt drei primäre Herausforderungen bestehender Arbeiten in Angriff: mangelnde Ausdruckskraft beim Benchmarking paralleler DSP-Workloads, die Notwendigkeit heterogener Hardwareunterstützung und die Notwendigkeit der Integration gelernter DSP-Modelle. Im Gegensatz zu bestehenden Systemen ermöglicht PDSP-Bench die Evaluierung von parallelen DSP-Anwendungen und -Workloads sowohl mit synthetischen als auch mit realen Anwendungen und ist eine ausdrucksstarke und skalierbare Lösung. Darüber hinaus erleichtert es das systematische Training und die Evaluierung von gelernten DSP-Modellen auf verschiedenen Streaming-Workloads, was für die Leistungsoptimierung entscheidend ist. Die umfassende Evaluierung von PDSP-Bench demonstriert seine Benchmarking-Fähigkeiten und zeigt die Auswirkungen unterschiedlicher Abfragekomplexitäten, Hardware-Konfigurationen und Workload-Parameter auf die Systemleistung. Die wichtigsten Beobachtungen unserer Experimente zeigen die Nichtlinearität und die paradoxen Auswirkungen der Parallelität auf die Leistung.

Zweitens leistet diese Arbeit einen Beitrag zu Methoden der Leistungsvorhersage und -optimierung, indem sie ZeroTune vorschlägt, ein neuartiges Learned-Cost Modell für DSP-Workloads und einen Optimierer für die Parallelitätsabstimmung. Es liefert hochpräzise Kostenvorhersagen und lässt sich gleichzeitig auf (unbekannte) heterogene Hardware-Ressourcen in der Cloud verallgemeinern. Die Verallgemeinerbarkeit des Modells basiert auf *Transfer-Lernen*, der gleichen Technik, die in *Large Language Models* wie Chat-GPT verwendet wird. Die Hauptidee besteht darin, von sogenannten *übertragba-*

ren Merkmalen und einer *parallelen Graphdarstellung* zu lernen, die es dem Modell ermöglichen, sich auf unbekannte DSP-Workloads und Hardware zu verallgemeinern. Eine umfangreiche Evaluierung zeigt die Robustheit und Genauigkeit von ZeroTune bei verschiedenen Workloads, verschiedenen Parallelitätsgraden, unbekannten Operatorparametern und Trainingsdateneffizienz. Die Auswertungen zeigen signifikante Geschwindigkeitssteigerungen durch Parallelitätstuning im Vergleich zu bestehenden Methoden. Darüber hinaus wurde unser Ansatz von *Amazon Redshift* für die Vorhersage der Abfrageausführungszeit übernommen.

Acknowledgments

I would like to express my sincere gratitude to my advisor, Prof. Ralf Steinmetz, for the opportunity to pursue a doctoral degree at the Multimedia Communications Lab (KOM) at TU Darmstadt. His invaluable guidance and continued support were instrumental in the successful completion of this research. I am also deeply thankful to my second advisor, Prof. Boris Koldehofe (TU Ilmenau), for his constant supervision and encouragement throughout my work in the field of Stream Processing. My heartfelt thanks go to Prof. Carsten Binnig from the Data and AI Systems Lab and DFKI, Darmstadt, for his insightful research discussions and guidance, which greatly contributed to the key publications of this dissertation.

I am very grateful for the research opportunity within the Collaborative Research Center MAKI, funded by the DFG. As a research scientist in MAKI, I had the privilege of collaborating with and learning from many exceptional individuals, including Prof. Dr. Antonio Fernández, Dr. Maik Benndorf, Dr. Mikhail Fomichev, Dr. Pegah Golchin, Roman Heinrich, and Chengbo Zhou. I also gained much from exchanging ideas with the research associates across various MAKI sub-projects, particularly the members of the transition and demo groups.

I would like to extend my gratitude to all the members of MAKI, KOM, the DSOS group (TU Ilmenau), the Distributed Systems Group (University of Groningen), and the Data and AI Systems Group (TU Darmstadt) for their collaboration and support. Working alongside such talented and dedicated individuals has made this journey deeply rewarding. I am also thankful to all the present and past members of KOM for providing a friendly and supportive working environment. I would also like to thank my Master students whom I supervised, for their involvement in this work. My special thanks to Dr. Julian Zobel for his invaluable help during the final stages of my thesis, as well as his patience and support. Also, I would like to specially thanks the reviewers of my dissertation - Dr. Anam Tahir, Dr. Christoph Gärtner, Dr. Pegah Golchin, Dr. Ralf Kundel, Benjamin Becker for providing valueable feedback for improvement.

A special and heartfelt thanks go to my wonderful wife, Manisha Luthra Agnihotri, for her unwavering support and encouragement throughout this journey. My deepest gratitude extends to my brother, Utkarsh Agnihotri, and his wife, Mansi Kakkar, for their support and assistance in proofreading this dissertation. Finally, I am profoundly grateful to my parents, Mr. Rajendra Kumar and Mrs. Madhuleshmani Agnihotri, for their endless love, support, and encouragement. Thank you all for making this journey possible.

Author's Publications

Major Publications

The author has published significant portions of the content presented in this thesis in the following peer-reviewed publications:

[1] Pratyush Agnihotri, Boris Koldehofe, Paul Stiegele, Roman Heinrich, Carsten Binnig, and Manisha Luthra. "ZeroTune: Learned Zero-Shot Cost Model for Parallelism Tuning in Stream Processing." In: *Proceedings of IEEE 40th International Conference on Data Engineering (ICDE)*. 2024, pp. 2040–2053 (cit. on pp. xiii, xvii, 162, 172, 173, 176–181, 183).

[2] Pratyush Agnihotri, Boris Koldehofe, Roman Heinrich, Carsten Binnig, and Manisha Luthra. "PDSP-Bench: A Benchmarking System for Parallel and Distributed Stream Processing." In: *Proceedings of the Performance Evaluation and Benchmarking - 16th TPC Technology Conference, TPCTC*. 2024, pp. 1–22 (cit. on pp. xiii, xvi, xvii, 162–169, 171).

[3] Pratyush Agnihotri, Boris Koldehofe, Carsten Binnig, and Manisha Luthra. "Zero-Shot Cost Models for Parallel Stream Processing." In: *Proceedings of the Sixth International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. 2023, pp. 1–5 (cit. on pp. xiii, xvi, xvii, 172, 173, 176–180).

[4] Pratyush Agnihotri, Boris Koldehofe, Carsten Binnig, and Manisha Luthra. "PANDA: Performance Prediction for Parallel and Dynamic Stream Processing." In: *Proceedings of the 16th ACM International Conference on Distributed and Event-Based Systems*. 2022, pp. 180–181 (cit. on pp. xiii–xvii, 168, 169, 171).

[5] Pratyush Agnihotri, Manisha Luthra, Miguel Rodriguez, and Boris Koldehofe. "IoT-Opt: The Swiss Army Knife to Model and Validate the Performance of IoT Products: Demo Abstract." In: *Proceedings of the 23rd International Middleware Conference Demos and Posters*. 2022, pp. 13–14 (cit. on pp. xiii, xiv, xvi).

[6] Pratyush Agnihotri. "Autonomous Resource Management in Distributed Stream Processing Systems." In: *Proceedings of the 22nd International Middleware Conference: Doctoral Symposium*. 2021, pp. 19–22 (cit. on pp. xiii–xvi).

[7] Mikhail Fomichev, Manisha Luthra, Maik Benndorf, and Pratyush Agnihotri. "No One Size (PPM) Fits All: Towards Privacy in Stream Processing Systems." In: *Proceedings of the 17th ACM International Conference on Distributed and Event-Based Systems*. 2023, pp. 61–67 (cit. on pp. xiii, xv).

[8] Pegah Golchin, Chengbo Zhou, Pratyush Agnihotri, Mehrdad Hajizadeh, Ralf Kundel, and Ralf Steinmetz. "CML-IDS: Enhancing Intrusion Detection in SDN Through Collaborative Machine Learning." In: *Proceedings of the 19th International Conference on Network and Service Management (CNSM)*. Best Paper Award. 2023, pp. 1–9 (cit. on pp. xiii, xv).

Additional Publications

Additionally, the author contributed as a co-author to the following peer-reviewed publications, which are not included in this thesis:

[9] Pratyush Agnihotri, Manisha Luthra, and Sascha Peters. “UrbanPulse: Adaptable Middleware to Offer City and User Centric Smart City Solution.” In: *Proceedings of the 20th International Middleware Conference Demos and Posters (Middleware)*. 2019, pp. 29–30.

[10] Pratyush Agnihotri, Florian Weber, and Sascha Peters. “Axxessity: City and User-Centric Approach to Build Smart City Bonn.” In: *Proceedings of the 13th ACM International Conference on Distributed and Event-Based Systems*. 2019, pp. 220–223.

[11] Manisha Luthra, Sebastian Hennig, Pratyush Agnihotri, Lin Wang, and Boris Koldehofe. “Highly Flexible Server Agnostic Complex Event Processing Operators.” In: *Proceedings of the 20th ACM/IFIP International Middleware Conference: Posters and Demos (Middleware)*. 2019, pp. 11–12.

[12] The An Binh Nguyen, Pratyush Agnihotri, Christian Meurisch, Manisha Luthra, Rahul Dwarakanath, Jeremias Blendin, Doreen Böhnstedt, Michael Zink, and Ralf Steinmetz. “Efficient Crowd Sensing Task Distribution Through Context-Aware NDN-Based Geocast.” In: *Proceedings of the 42nd IEEE Conference on Local Computer Networks*. 2017, pp. 52–60.

Previously Published Material

This thesis contains material that has been previously published in various scientific conferences. Table 1 summarizes the relationship between the former publications and the chapters of this thesis. None of the text in this thesis is directly taken from the publications. However, figures, tables, algorithms, notations, and results that exclusively represent the core concepts, have been used consistently in this thesis to prevent discrepancies with the true data that is formerly published. The list of all the author's scientific publications is available in the previously presented *Chapter Author's Publications*.

Section	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
Chapter 2: Fundamentals and State of the Art								
Parallel and Distributed Stream Processing	✓	✓		✓		✓	✓	✓
Performance Modeling Methods	✓	✓				✓	✓	✓
Chapter 3: Performance Model Architecture								
Scenario Description				✓	✓	✓	✓	✓
Conceptual Architecture				✓	✓	✓	✓	✓
Chapter 4: PDSP-Bench for Performance Benchmarking								
Benchmarking System Analysis		✓		✓	✓	✓		
PDSP-Bench *Design*		✓		✓				
Chapter 5: ZeroTune for Performance Prediction								
Cost Model Design	✓		✓	✓				
Data-Efficient Training	✓	✓						
Prediction and Optimization	✓							
Chapter 6: Evaluation								
Real-world Evaluation Setup	✓	✓	✓	✓	✓			
Performance Benchmarking		✓		✓				
Accuracy and Generalizability	✓		✓	✓				
Appendix B								
Resource Usage Evaluation		✓		✓				
ZT-Random Model Evaluation	✓		✓	✓				

Table 1: List of previously published and peer-reviewed scientific publications used in the corresponding chapters of this thesis.

This thesis represents the scientific work carried out in the scope of DFG (German Research Foundation) funded Collaborative Research Center 1053 MAKI (Multi-Mechanism Adaptation for the Future Internet)[1]. MAKI offers an interdisciplinary collaboration research work spanning computer science, electrical engineering, information technology, and economics, aiming to develop a highly adaptive and efficient Future Internet. Therefore, the publications included in this thesis are the product of collaborative efforts among researchers from the departments of computer science, electrical engineering, and economics, many of whom are affiliated with either the MAKI project or the Multimedia Communications Lab (KOM) at the Technical University of Darmstadt. In this chapter, the pronoun "I" details my specific contributions to each publication. Contributions from co-authors, along with their affiliations, are acknowledged accordingly. Co-authors not linked to a specific institution are my colleagues from the Multimedia Communications Lab or are involved with the MAKI project. Throughout the rest of this thesis, the pronoun "we" is employed to refer to all co-authors of the cited works collectively.

The contributions provided in this thesis were continually supervised by Prof. Dr. Boris Koldehofe (Professor at the Technical University of Ilmenau), Prof. Dr. Carsten Binnig and Prof. Dr.-Ing. Ralf Steinmetz. I have discussed their concrete contributions in the description below. However, wherever not explicitly mentioned, they generally contributed via continuous supervision.

Chapter 2, *Fundamentals and State-of-the-art*, presents the foundational concepts and existing literature related to performance modeling in parallel and distributed stream processing. I developed the core concepts, identified the requirements for parallel and distributed stream processing, and conducted a thorough analysis to pinpoint the fundamental research gaps and challenges that this work aims to address. I published identified research gaps and the resulting challenges in designing such performance modeling and parallelism tuning for DSP systems in [6].

Chapter 3, *Performance Modeling and Parallelism Tuning*, outlines a common DSP scenario that spans various application domains illustrating the broad applicability and need of the proposed solution in this thesis. This scenario description is followed by a comprehensive overview of the system model and logical architecture of the proposed system for performance modeling and parallelism tuning in DSP systems. These presented concepts and overall performance modeling models and methods were discussed and developed with Prof. Binnig, Prof. Koldehode, and Dr. Luthra. These findings and initial results were later published in [4, 5, 6]

[1] MAKI Multi Mechanism Adaptation for the Future Internet `https://www.maki.tu-darmstadt.de/` [Accessed in 25.04.2024].

Additionally, I collaborated with Pegah Golchin and Chengbo Zhou to explore the practical application of my proposed models for accelerating performance using machine learning methods and hardware accelerators. We supervised lab [1] and Mater thesis [1], focusing on real-time in-network traffic intrusion detection to enhance network security. I contributed by providing input on conceptual design and machine learning models, as well as improving evaluation and applicability in real-world scenarios. The results of this joint work were published in [8], earning the *Best Paper Award* at IEEE for our contributions.

Furthermore, I investigated the direction of applying various optimization mechanisms, e.g., placement and parallelism tuning, to achieve Quality of Service (QoS) requirements while real-time data processing on resources such as cloud, fog, and edge nodes. I collaborated with Dr.-Ing. Mikhail Fomichev (Seemoo, TU Darmstadt), Dr.-Ing. Manisha Luthra (DFKI and TU Darmstadt) and Dr. Maik Benndorf (University of Oslo). In this joint work, I contributed by defining DSP processing scenario in the context of preserving users' privacy, followed by designing and conceptualizing to apply various optimizations such as operator placement, operator parallelization as well and privacy-preserving mechanisms (PPM) in DSP systems. Later, this joint work was published in [7], where all the co-authors contributed to the publication's writing.

Chapter 4, building on my previous investigation and identified gaps [6, 4], I designed and developed PDSP-BENCH benchmarking system to facilitate a systematic understanding of performance and resource configurations *for Parallel and Distributed Stream Processing*. I contributed by investigating existing performance modeling methods and the framework required to build such a system, followed by identifying the challenges of building a benchmarking system that can facilitate understanding and analyzing performance and generate a corpus of performance data for DSP system optimization. Furthermore, I conceptualized the overall system design, which included modeling performance metrics, adapting and implementing real-world and synthetic applications, identifying workload- and resource-specific parameters, and conducting preliminary evaluations to understand their influence on performance across heterogeneous cloud resources and evaluation of well-known DSP systems such as Apache Flink, Apache Storm, and Heron.

I extended the idea of a benchmarking system for DSP systems and extensively evaluated the performance of various data flows for varying hardware configurations and workload characteristics for different performance metrics such as end-to-end latency and resource utilization. Additionally, I integrated the learned cost models into the benchmarking system to offer a fair method to collect data for performance analysis. I also used the same data to train machine learning models for performance prediction as well as DSP system optimization such as parallelism tuning and placement. Subsequently, I have published the overall idea of PDSP-Bench, a novel performance modeling using benchmarking system [2]. All the co-authors of the aforementioned works [6, 5, 2] contributed to the written text of the manuscript.

Additionally, I explored how different industries use performance benchmarking to demonstrate the real-world applicability of the solution. For this, I collaborated with Miguel Rodriguez (Senior Program Manager, Deutsche Telekom) to further investigate different IoT devices and their performance with varying application scenarios, network and hardware configuration. I collaborated with extending the existing solution of IoT Solution Optimizer with axxessio GmbH to include an additional module for benchmarking systems. We published the overall system design, identified challenges, and initial evaluation results in [5], where all the authors contributed in writing the paper. We presented our industry solution, which efficiently models and validates the performance of proposed or deployed IoT products quickly.

Chapter 5 introduces ZeroTune, a novel performance prediction model for DSP that determines initial parallelism degrees based on predicted performance without query execution, significantly reducing the need for costly adjustments during the early stages of query execution. I conducted a thorough literature review on the existing analytical and machine learning methods for optimizations in DSP systems, such as parallelism, placement, or resource provisioning based on workload and resource characteristics. Following this, I developed a basic learning cost model and conducted a preliminary evaluation, which was essential to identifying the research gap and challenges to designing a performance model that can accurately predict performance and be generalizable across various workload and resource characteristics that are not even part of the training data. These findings were detailed in our publication [4, 3].

I also investigated various tools and frameworks necessary for implementing such solutions, notably using Apache Flink for developing workload generators for generating diverse training and testing data sets and setting up metal servers on CloudLab to simulate real-world DSP system deployments. This exploration also covered the foundational concepts of Zero-shot learning, including necessary features and their representations. Subsequently, I supervised the M.Sc. thesis of Paul Stiegele [2] with Dr. Luthra and Prof. Boris. Using transfer learning, he developed a proof of concept, focusing on model accuracy against test data. In addition, Ph.D student Roman Heinrich (DFKI Darmstadt) of Prof. Binnig and Dr. Luthra, who are working in the direction of transfer learning for operator placement, also time-to-time shared his insights about his work to improve the proof of concept further. Furthermore, I continued to extend and conceptualize the idea of generalizability by reducing training time and data. I extended the proof of concept to compare in addition to comparison with different baselines and sophisticated machine learning architectures. These contributions were later published in [1]. All co-authors provided invaluable feedback and contributed significantly to the refinement and review of the manuscript.

Chapter 6 presents a comprehensive evaluation of the proposed contributions to assess the effectiveness and applicability of performance modeling approaches in real-world scenarios across diverse streaming workloads and computational environments provided by cloud resources. Building on the proof of concept from the master's theses [2, 3], I extended the evaluation framework to include further specialized workload and resource configurations designed to reflect the dynamic and heterogeneous nature of modern DSP operations, making it an essential tool for comparative analysis and a deeper understanding of different modeling approaches. A significant focus of my evaluation centers on ZEROTUNE, where I extensively evaluate and analyze the accuracy with which performance models, particularly in terms of predicting costs such as latency and throughput, can generalize and adapt across varied DSP workloads with minimal training, including different query types and data streams, and resource configurations.

Additionally, I have expanded the evaluation of PDSP-BENCH to systematically explore the performance of real-time and synthetic queries across various DSP applications, encompassing diverse workloads and heterogeneous hardware setups. This also includes integrating machine learning models to leverage the data generated for model training. The insights gained from these comprehensive evaluations have been published [4, 3, 1, 2], which detail the significant findings and contributions of our research. Prof. Koldehofe, Dr. Luthra, and Prof. Binnig have provided valuable feedback on evaluation results to improve further and examine the performance models.

Finally, this thesis was written entirely by the author, and no content was generated by AI. AI tools such as Grammarly and ChatGPT are only used for grammar and spelling correction and improving inconsistency in sentence structure.

Content

1

Introduction and Motivation

There has been a significant rise in modern data-driven applications across various sectors, e.g., financial trading [203], social media and entertainment trend monitoring [2], healthcare [247], traffic monitoring [81], and fraud detection [160]. For instance, entertainment platforms like *Netflix* handle approx. 1.3 TB data daily [33] to deliver seamless streaming experiences, while social media platforms like *Facebook* [139] process 9 GB user data per second to analyze user interactions in real-time and provide personalized content. One of the critical requirements for these data-driven applications is *timeliness*, where real-time data processing and response are essential.

Rise of data-driven applications...

Distributed Stream Processing (DSP) systems have emerged as a critical technology for the growing demands of these applications. For instance, Netflix uses *Apache Flink* [33] and Facebook uses *Puma* stream processing engines [58]. To meet the real-time and high throughput requirements of these applications, DSP systems offer a *dataflow* abstraction that chains the compute units called *operators* in a so-called *dataflow graph*.

...necessitates use of DSP system...

Current DSP systems often deploy dataflow graphs on cloud resources to meet the performance requirements of these modern data-driven applications. This is because cloud platforms provide the necessary computational power and scalability to handle massive data volumes and complex processing tasks. However, cloud resources are inherently *heterogeneous*, which makes understanding the performance of DSP workloads very difficult. This is because the performance highly depends on the available resources – compute, storage, and network – all being heterogeneous, making the problem even harder. Moreover, the colocation of DSP workloads on the same hardware further complicates the understanding [267]. For instance, Amazon EC2[2] provides different instances optimized to fit for different use cases. There are instances that are optimized for computing (CPUs), memory (RAM), and storage (SSDs, disk space) separately. For instance, memory-optimized instances (R7g instances) offer nodes ranging from memory sizes of $8 - 512$ GiB.

...on the cloud.

Hard to assess DSP in heterogeneous environments

[2] Web service of Amazon provides a wide selection of cloud instances. `https://aws.amazon.com/ec2/instance-types/` [Accessed on 25.06.2024]

Parallelism is a desired property but...

To deal with the high workload requirements of current data-driven applications, DSP systems specify the operator *parallelism* in the graph specification and provide data partitioning strategies to manage these parallelised dataflows on the cloud. While such parallel dataflows have become an intrinsic part of every DSP system, there is, however, no means of systematic understanding of DSP performance under massively parallel dataflows as most of the existing systems [20, 64, 250, 132] are tailored towards the understanding of sequential dataflows.

...existing systems are limited in performance modeling of parallel dataflows.

There exists benchmarking systems for DSP [238, 91, 42, 111] that limitedly assess parallel dataflow applications. However, they are restricted to homogeneous hardware. In the real world, applications like Netflix run on $1400+$ nodes on $50+$ distinct clusters with varied CPU cores [33] to deal with their demands of massively parallel dataflow applications. Therefore, accurate performance modeling for DSP is essential to solve these challenges and ensure that applications can consistently meet their performance goals, e.g., timeliness. DSP systems can use accurate performance modeling to predict how different configurations will perform, enabling informed decisions to determine operator parallelism or operator placement, i.e., the deployment of parallel operators on heterogeneous hardware.

Predicting performance of parallel dataflows is even harder

However, predicting the performance of parallel dataflows on heterogeneous hardware is a very challenging problem. This is because the application workload is very dynamic, and it changes with time. For instance, in the *Netflix* example, movie requests vary depending on the time of the day, with more requests in the evening after office hours while fewer requests during the day. Moreover, the workloads are not known in advance, and the prediction model must adapt to the kind of workload for which it does performance prediction. Above all, executing the workload on heterogeneous hardware and their unpredictable behavior in the cloud, as discussed before, make the problem even more difficult [267, 139].

This thesis contributes to accurate performance modeling for distributed stream processing in heterogeneous environments by proposing a benchmarking system and zero-shot machine learning models. The results can be used to perform mechanism *transitions*. Mechanism transitions is a concept established in the research project Collaborative Research Centre “MAKI” [15, 254, 88, 164, 196] where adaptations between mechanisms are performed to meet the performance requirements of the underlying system. By leveraging the results established in this thesis, effective transitions between parallelism mechanisms of DSP system can be performed based on the predicted performance of zero-shot cost models. In the following, we discuss the research challenges and goals of this thesis.

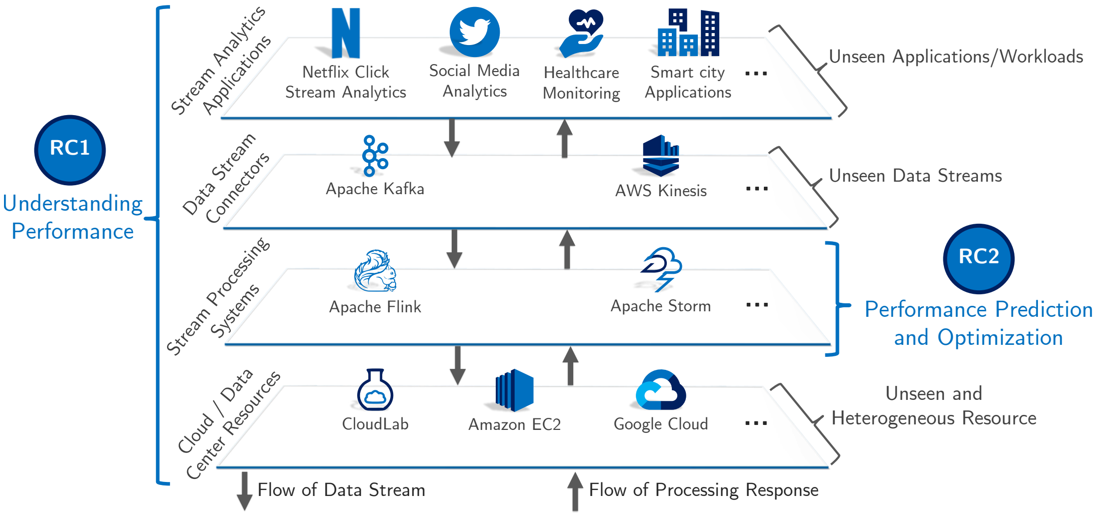

Stream Analytics Applications
Netflix Click Stream Analytics
Social Media Analytics
Healthcare Monitoring
Smart city Applications
...
Unseen Applications/Workloads
Data Stream Connectors
Apache Kafka
AWS Kinesis
...
Unseen Data Streams
RC1
Understanding Performance
Stream Processing Systems
Apache Flink
Apache Storm
...
RC2
Performance Prediction and Optimization
Cloud / Data Center Resources
CloudLab
Amazon EC2
Google Cloud
...
Unseen and Heterogeneous Resource
Flow of Data Stream
Flow of Processing Response

Figure 1: Overall system architecture showing the two research challenges solved in this thesis on RC1: performance understanding and RC2: performance prediction and optimization.

1.1 Research Challenges

The distributed and heterogeneous environments of stream processing impose significant challenges for performance understanding and prediction of DSP workloads, as discussed above. Based on this, we elaborate on these two core research challenges (RC). Figure 1 positions the research challenges in an overall architecture that we propose in this thesis comprising of DSP applications, data stream connectors, DSP systems and cloud providers offering heterogeneous resources.

RC1: *Understanding the performance of parallel dataflows in heterogeneous environments.*

DSP on heterogeneous environments is hard to assess...

...under varying conditions.

In the context of DSP systems, understanding the performance of DSP workloads within heterogeneous environments presents a significant challenge. Firstly, heterogeneous cloud environments encompass a wide range of hardware configurations, from high-performance to low-power resources, each with varying CPU speeds, memory sizes, and I/O capabilities. Secondly, network conditions such as latency and bandwidth can vary unpredictably, affecting the efficiency of data transfer and synchronization between distributed operators. This diversity makes it difficult to generalize performance metrics, as the behavior of DSP operations can differ significantly depending on the underlying hardware. Moreover, the fluctuating availability of cloud resources adds another layer of complexity. Additionally, DSP systems must handle a wide variety of data from different applications with unique QoS requirements. Above all, parallel dataflows behave very differently than sequential dataflows because both the number of parallel instances of operators and data partitioning among them have a huge influence on the performance of DSP systems. Moreover, incorrect provisioning of parallel operators results in effects like backpressure (reduced data rate); therefore, finding a good balance and optimal parallelism degrees for operators requires detailed performance benchmarking. Together, these factors create a highly complex and variable environment, making it difficult to explore potential configurations and scenarios for performance modeling for parallel dataflows, necessitating advanced, comprehensive solutions to ensure consistent and efficient DSP system performance.

RC2: *Performance prediction and optimization under dynamic workloads and heterogeneous environments.*

In the first challenge, we highlighted the importance of performance benchmarking to find out optimal parallelism degrees for parallel dataflow applications. However, performance benchmarking is expensive; in fact, executing DSP workloads on a testbed like CloudLab [76] requires several days or even weeks to benchmark performance depending upon the number of resources. The use of machine learning (ML) for performance prediction of DSP systems is a viable option and existing work adjusts parallelism [213, 140, 201] or uses ML methods to provision parallel resources based on the predicted costs [112, 67, 54]. A key challenge, however, for such approaches is their applicability for different DSP workloads and executing them on heterogeneous environments. This is especially difficult because current approaches learn a model specific to a DSP workload. For instance, a model that learns the performance of queries specific to Netflix fails to generalize to queries specific to Facebook. Existing ML models train specifically for each workload (DSP data and query), which is clearly not a scalable solution. Another problem is that current ML models hardly take heterogeneous hardware into account in their featurization [174, 131, 25]. Even when they do, they suffer from similar generalization problems for unseen hardware [155, 249] like for unseen workloads, as explained above. Clearly, this is a problem for DSP workload execution on heterogeneous cloud resources.

Benchmarking is expensive...

...to explore all viable configurations.

Existing models are workload-specific...

...and lack generalization.

1.2 Research Goals and Contributions

We show that the research challenges found in the above section are solved by the contributions of this thesis: *(i)* PDSP-BENCH: A benchmarking system for parallel and distributed dataflow applications of DSP and *(ii)* ZEROTUNE: A zero-shot cost model that predicts precise performance of parallel dataflow applications even when they execute on heterogeneous hardware. Thus, the main goal of this thesis is to provide methods for accurate performance mod-

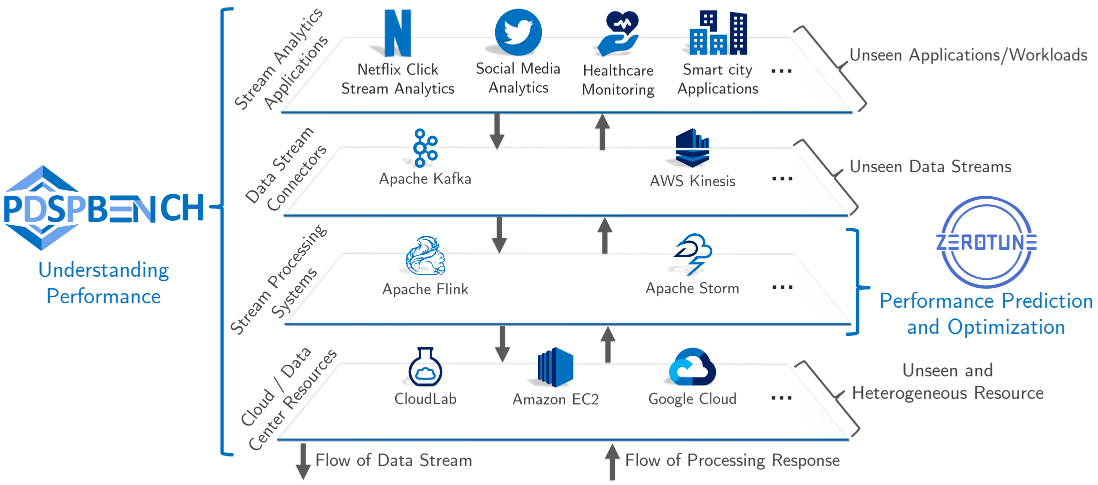

PDSPBENCH
Understanding Performance
Stream Analytics Applications
Netflix Click Stream Analytics
Social Media Analytics
Healthcare Monitoring
Smart city Applications
...
Unseen Applications/Workloads
Data Stream Connectors
Apache Kafka
AWS Kinesis
...
Unseen Data Streams
Stream Processing Systems
Apache Flink
Apache Storm
...
ZEROTUNE
Performance Prediction and Optimization
Cloud / Data Center Resources
CloudLab
Amazon EC2
Google Cloud
...
Unseen and Heterogeneous Resource
Flow of Data Stream
Flow of Processing Response

Figure 2: The proposed contributions in this thesis (1) PDSP-BENCH [7]: A benchmarking platform for parallel and distributed dataflow applications of DSP for *RC1* and (2) ZEROTUNE [8]: a zero-shot cost model that predicts accurate performance of parallel dataflow applications even when they execute on heterogeneous hardware to tackle *RC2*.

eling as follows: *(i)* By benchmarking parallel dataflow applications of DSP using our PDSP-BENCH system, and *(ii)* By accurate performance prediction of parallel dataflow applications across heterogeneous hardware using our generalizable cost model named ZEROTUNE. Thus, we divide into the following two main research goals, one for each contribution.

Research Goal 1: *Systematic performance benchmarking of parallel and distributed stream processing workloads in heterogeneous environments.*

This research goal focuses on a systematic understanding of the performance of parallel dataflows of DSP applications under heterogeneous cloud environments. For this, we propose PDSP-BENCH, the first benchmarking system that enables the creation and evaluation of parallel dataflow graphs and provides large corpora based on 23 synthetic and real-world applications on heterogeneous testbeds of CloudLab resources. Using these applications as workload structures for parallel dataflows, we generate several thousand queries that we benchmark using PDSP-BENCH.

PDSP-BENCH *for performance benchmarking...*

By facilitating the integration of this large variety of parallel dataflow workloads and support for the management of heterogeneous hardware resources from Cloudlab [76], we provide the first benchmarking system for parallel DSP systems. In addition to benchmarking, we also show ML models can be trained on the generated workloads using PDSP-BENCH, which allows a fair comparison between the ML models by generating consistent training

...with data-generation for ML.

data. We evaluate PDSP-Bench by using a widely used DSP system, *Apache Flink* [52, 50] as a System Under Test (SUT) and demonstrate the impact of varying parallelism complexities, hardware configurations and DSP workload parameters on DSP system performance.

Research Goal 2: *Generalizable and data-efficient learned performance models for DSP in heterogeneous environments.*

ZeroTune *for accurate performance prediction...*

This research goal focuses on overcoming the challenge of accurate performance prediction of distinct parallel dataflow applications on heterogeneous hardware. For this, we propose ZeroTune, a novel zero-shot cost model that gives precise predictions on the performance (e.g., latency and throughput) of executing parallel dataflow graphs on heterogeneous hardware resources. The zero-shot property of ZeroTune means that we prevent the need for constant retraining of the model for every new parallel dataflow workload encountered for the inference or when new hardware becomes available in the cloud.

...and generalization for unseen dataflow...

...with parallelism tuning.

To this end, we propose a novel cost model that allows DSP to tune initial parallelism degrees without the need of executing any workloads on DSP systems. The model is based on a graph neural network (GNN), and we carefully select transferable features that allow the model to generalize to unseen parallel dataflow structures and hardware. For parallelism-related training data, we propose a data-efficient training data collection strategy *OptiSample*. Unlike the random selection of parallelism, OptiSample determines the right number of processing instances for each operator using analytical approaches [225, 83, 131] that allows high accuracy of the learned model in fewer workloads and training time. An exhaustive evaluation confirms ZeroTune's robustness and accuracy of ZeroTune across a wide spectrum of workloads, including seen and unseen scenarios, across varying degrees of parallelism, and in the context of unseen operator parameters. Furthermore, it highlights the model's data efficiency during training and the significant performance improvements achieved through its parallelism tuning strategy compared to alternative methods.

Open Source Contributions

We have released the software artifacts related to our contributions PDSP-Bench and ZeroTune as open-source software under Apache License 2.0.

- `https://github.com/pratyushagnihotri/PDSPBench`: This repository contains the source code of PDSP-Bench for performance benchmarking [7]. To this end, the repository includes step-by-step documentation for cloud environment setup, initiating the benchmarking process, performance monitoring, data generation, benchmarking prediction models, and benchmark results.

- `https://github.com/pratyushagnihotri/ZeroTune`: This repository contains the source code of ZeroTune zero-shot performance prediction model with optimizer for initial parallelism tuning [8]. To this end, the repository includes step-by-step documentation for distributed environment setup, training data generation, training, and inference using learned performance models, as well as an optimizer for initial parallelism tuning.

1.3 Structure of the Thesis

Following this introduction and motivation for our research challenges, goals, and contributions, *Chapter 2* provides a conceptual background on parallel and distributed stream processing and performance modeling methods, which are essential to understand the core of the DSP system. *Chapter 3* discusses the common scenario, system model, and an overview of the contributions. *Chapter 4* presents performance benchmarking system PDSP-Bench, its components and the benchmarking workflow. *Chapter 5* presents the zero-shot cost model ZeroTune, including the feature selection, efficient data generation using training strategies, the training and inference process, and an optimizer for parallelism tuning. *Chapter 6* presents an extensive evaluation of PDSP-Bench and ZeroTune on different workloads, cloud hardware resources, and their varying configurations as well as their comparison with existing baselines to show performance benchmarking capability, prediction accuracy, and generalization of proposed methods. Finally, *Chapter 7* concludes this thesis with a summary of our contributions and an outlook on potential future directions.

2

Fundamentals and State of the Art

In this chapter, we lay the groundwork by offering foundational insights into the concepts and an overview of the state of the art to understand the contributions discussed in this thesis. Our discussion commences with a deep dive into the concept of parallel and distributed stream processing, the cornerstone of our thesis, and its associated components will be thoroughly explained in Section 2.1. Following this, our focus shifts to performance modeling of DSP, exploring the various methods available. We examine diverse measurement approaches, including learned and non-learned methods, and discuss how these methods can be effectively applied to performance modeling and initial parallelism tuning in Section 2.2. Finally, we conclude this chapter by presenting a comprehensive overview of existing research related to DSP, specifically focusing on methods for performance modeling, parallelism, and resource provisioning in Section 2.3.

2.1 Parallel and Distributed Stream Processing

The widespread adoption of digital devices and tools has led to a rapid growth in the amount, diversity, and speed of data [4, 198, 47, 164]. This expansion has led to the creation of vast pools of data across various applications, spawning from various sources such as *click analytics* [171], *health monitoring* [212], *financial transactions* [85], and so on. In the current digital landscape, timeliness and high workload processing have become critical factors in these modern data-driven applications. For instance, emerging applications in smart cities, banking, retail, infrastructure monitoring, and Internet of Things (IoT) require the rapid processing of continuous data streams to ensure minimal latency and maximum throughput [234, 77]. The need for robust, real-time data processing solutions has never been more pronounced as industries strive to detect patterns, identify failures, and derive actionable insights from these data streams.

Timeliness is an essential factor for...

...better services in many real-world applications.

The rapid growth of data and the need for timeliness for data processing has precipitated the development and evolution of systems capable of processing data in real-time, known as Distributed Stream Processing (DSP) systems [47,

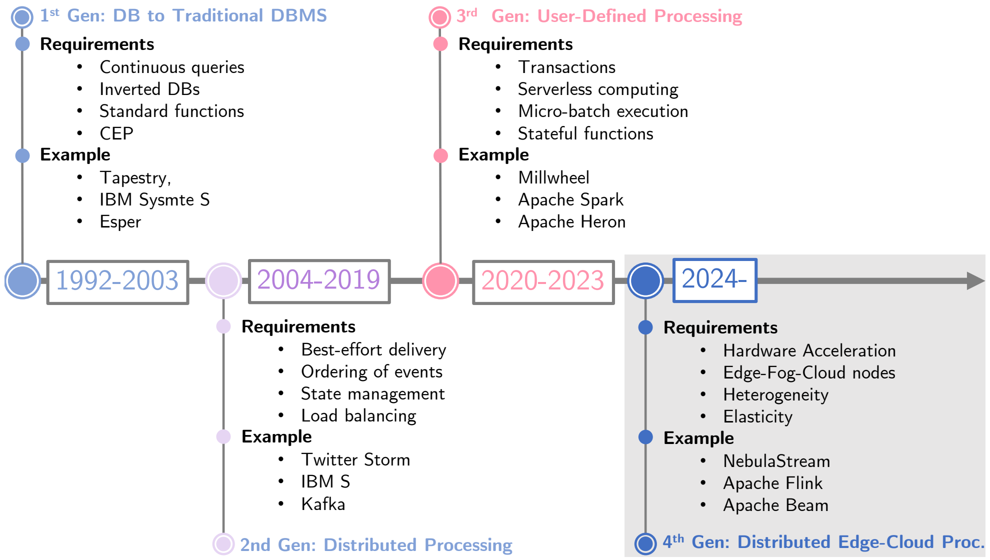


Figure 3: Evolution of DSP systems (adapted from [66, 51]).

185, 41, 207]. The evolution of DSP systems, depicted in Figure 3, reflects a journey from managing dynamic data streams to embracing the complexities of distributed, real-time processing across various architectures, including cloud and edge computing [24, 51, 198, 66, 138]. Initially, Data Stream Management Systems (DSMSs) adapted traditional database management systems to accommodate unbounded data streams, offering SQL-like languages for constructing long-running queries [145, 184, 47]. We are now witnessing the advent of a fourth generation of DSP systems, representing the forefront of stream processing methods, emphasizing on *user-defined operations*, explore *parallel processing* [173, 44, 4] and *resource elasticity*, and accelerate performance using *hardware-acceleration* [43, 147] to meet the challenges of timeliness in distributed stream processing in cloud computing environments [94].

Fourth generation of DSP is about...

DSP systems function as middleware [4, 198, 31, 47], processing data from various *data producers or sources* and delivering the results to *data consumers or sinks*. As illustrated in Figure 4, a DSP system can receive data from diverse *data producers* such as user tweets, sensor data from autonomous vehicles, gaming data, and financial transactions. The DSP system then continuously processes this data in response to queries from *data consumers*, such as banks or retail industries. It allocates the data across one or more computing *physical resources* for processing and extracting meaningful insights. For instance, it can analyze tweet data to *gauge user behavior, detect fraudulent transactions*, or generate targeted advertising recommendations based on gaming analysis. Ultimately, it delivers these insights to the *data consumers* in response to queries. Given this foundation, Definition 2 [66, 198] explains a distributed stream processing system as used in this thesis.

...parallel processing and hardware acceleration.

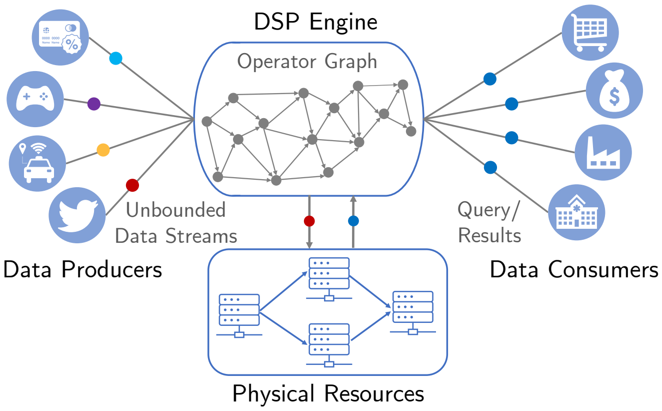


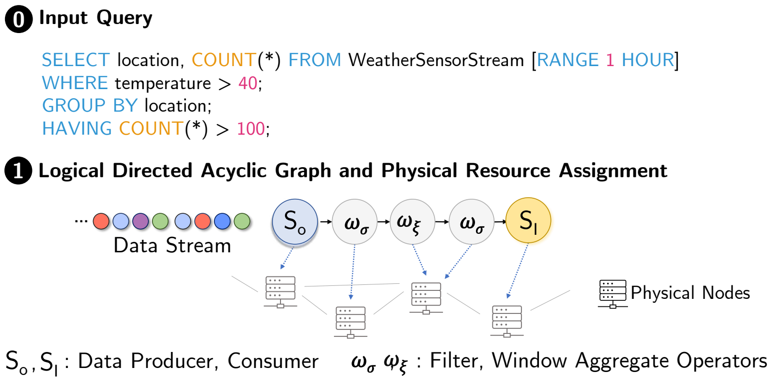
Figure 4: Overview of a DSP systems where an unbounded sequence of *data streams* produced from various *data producers* are processed step by step in *graphs of operators* based on *query* from *data consumers*, e.g., click stream analytics [33], financial banking [85]. After query processing, *results* are delivered to consumers in the form of finding complex patterns and triggering actions in real-time.

Definition 1. *Distributed Stream Processing (DSP) Systems*
DSP systems process continuous data streams across multiple computing resources or nodes in a network. This enables real-time data analysis by partitioning data streams and distributing workloads among multiple nodes.

To better understand this definition, let us consider the example of processing *weather sensor data* in a distributed manner, as presented in Figure 5. A `weather monitoring consumer` aims to identify locations with temperatures exceeding 40 degrees. For this, a consumer constructs ⓪ a query to analyze the `WeatherSensorStream`. Figure 5 illustrates a typical processing of data streams in a DSP system, where ⓪ represents an input query and ① presents a logical plan in the form of a Directed Acyclic Graph (DAG) that models the data flow from data producers (S_O) to consumers (S_I). In this DAG, *vertices* symbolize operators, e.g., filter (ω_σ), window aggregation (ω_ξ) that sequentially process streams to transform events into more complex outcomes progressively and *edges* denote the flow of data streams, processed by these operators. Moreover, the *physical plan*, or the assignment of physical resources, demonstrates how operators are deployed on cloud resources to process data streams while adhering to the QoS requirements specified by the consumers. These QoS requirements are generally specified in the queries and are met through the resource management strategies implemented within DSP systems.

DSP system offers...

...real-time processing of data.



```
SELECT location, COUNT(*) FROM WeatherSensorStream [RANGE 1 HOUR]
WHERE temperature > 40;
GROUP BY location;
HAVING COUNT(*) > 100;
```

1 Logical Directed Acyclic Graph and Physical Resource Assignment
...
Data Stream
S_o
ω_σ
ω_ξ
ω_σ
S_I
Physical Nodes
S_o, S_I : Data Producer, Consumer
ω_σ ω_ξ : Filter, Window Aggregate Operators

Figure 5: Example of DSP of weather sensor data [4, 8].

Parallel Stream Processing (PSP)

PSP enables multiple instances of operators...

DSP systems employ parallel stream processing (PSP) to achieve fast response time even under high workloads, allowing entire or sub-parts streams to be processed by multiple instances of an operator(s), which are placed either on a single node or distributed across multiple nodes [198, 159, 66] on cloud infrastructure. The concept of parallelism in streaming systems is often referred to as the *parallelism degree*, which specifies the number of operator instances required for data stream processing.

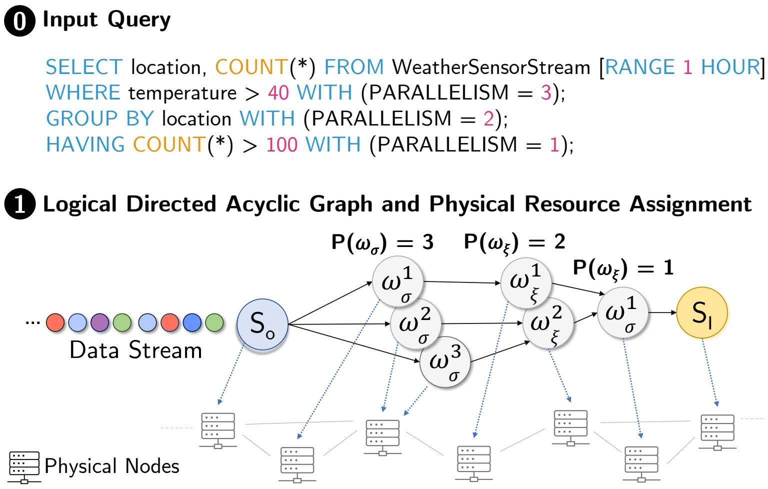
> **Definition 2.** *Parallel Stream Processing (PSP)*
> Parallel stream processing refers to the method of analyzing and processing data streams concurrently by multiple instances of operators across single or multiple computing resources or processors.

...to process a high amount of data in parallel.

In Figure 6, we show how parallelism degree is specified for each operator of ⓪ input query in DSP systems Taking our previous example (cf. Figure 5), multiple instances might utilize the *location ID* to *filter* data, which could then be *aggregated* by several instances of a *window aggregation* function. ① The physical plan details how these operators and their instances are allocated across the same or different nodes for processing on cloud resources. DSP systems use the logical term *parallelism degree* to define the number of parallel instances of an operator required for data stream processing. It plays a crucial role in how effectively a DSP system can process large volumes of data in real-time. Different configurations of parallelism can lead to various performance outcomes.

Inaccurate parallelism may...

For instance, Netflix [33] uses a stream processing system to perform live *click streams analytics* based on users' interactions with video content and insert real-time ads based on viewer demographics and content themes. The DSP system is configured with a low parallelism degree, assuming the



```
SELECT location, COUNT(*) FROM WeatherSensorStream [RANGE 1 HOUR]
WHERE temperature > 40 WITH (PARALLELISM = 3);
GROUP BY location WITH (PARALLELISM = 2);
HAVING COUNT(*) > 100 WITH (PARALLELISM = 1);
```

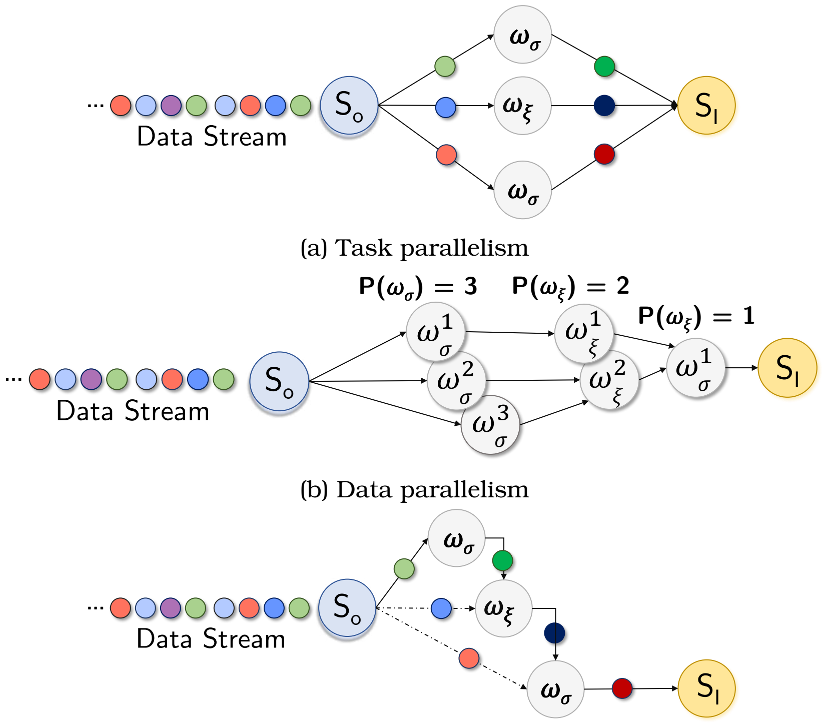
Figure 6: Example of PSP of weather sensor data where multiple instances of operators (filter, window aggregation), are distributed across cloud resources for processing [4, 8].

workload will be relatively light. During major live events (e.g., sports finals, live concerts), the viewer counts, and interaction rates surge unexpectedly. With few operator instances handling the increased workload, the majority of the computational resources of the system remain *idle*, i.e., *under-utilization of available cloud resources*, resulting in the slow processing of viewer data streams, and delayed ad insertions can cause potential revenue loss as ads fail to target viewers in optimal times. The underutilization reflects wasted computational capacity and missed opportunities for targeted advertising. Similarly, the traffic volume or workload significantly decreases during off-peak hours, such as late nights or early mornings. In such a scenario, despite the low data volume, the high number of operator instances consumes excessive computational power and memory, leading to high operational costs and system strain. The *over-utilization* results in unnecessary energy consumption and increased damage to hardware resources, escalating maintenance and operational costs without any real benefit. To tackle such performance deficiencies of DSP system, this thesis addresses the challenges associated with accurate performance modeling to determine the initial optimizations, e.g., parallelism degree as mentioned in Section 1.2 for parallel stream processing in DSP systems.

...result in inferior performance.

Types of Parallel Stream Processing

In DSP systems, parallel processing of operators can be categorized into three primary types (cf. Figure 7): *(i)* task, *(ii)* pipeline, and *(iii)* data parallelism [198, 159, 66]. These forms of parallelism represent different approaches to dis-



(a) Task parallelism

(b) Data parallelism

(c) Pipeline parallelism

Figure 7: Example of different parallelism strategies in DSP systems. (a) Task parallelism processes multiple operations on the same data stream in parallel; (b) Data parallelism, where multiple instances of operators process entire or sub-part of data streams; and (c) Pipeline parallelism, which processes the data stream sequentially in stages where the successor operator receive output from predecessor operator as input. Optionally, operators can also receive a full data stream in addition to the output stream from previous operators.

tributing the workload across operators and their instances processing on multiple computing resources to enhance performance and efficiency.

Data parallelism is...

...multiple instances of same of operators.

This dissertation primarily explores the concept of *data parallelism* to validate the proposed contributions, which defines the number of parallel instances of the same operator that processes a sub-part of a data stream (cf. Figure 7b). This approach is particularly effective for large-scale data processing, where it can significantly improve throughput and reduce processing time. However, the principles of developed methods can be adapted to *task* and *pipeline* parallelism with some fine-tuning. In *Task parallelism*, multiple operators are processed concurrently or in a pipeline way on the same input data stream (cf. Figure 7a), while *pipeline parallelism* organizes the processing stages into a pipeline, with each stage running in parallel on different data elements as they flow through the pipeline (cf. Figure 7c).

2.2 Performance Modeling for Distributed Stream Processing

Performance modeling in the context of this thesis refers to the systematic approach of understanding, analyzing, and predicting the behavior and performance of DSP systems under various conditions and configurations [79, 89]. The primary goal of performance modeling is to provide insights that enable efficient design [79, 89, 197], optimization [209, 164, 207, 78], and management [201] of DSP systems, ensuring they meet the stringent timeliness and quality of service (QoS) requirements of real-time data-driven applications. In the domain of performance modeling for DSP systems encompasses a range of methods, broadly classified into: *Non-learned* [66, 198, 24, 159, 130, 53, 152, 135] and *learned* methods [164, 217, 263].

Performance benchmarking...

Non-learned or Non-machine learning (Non-ML) consists of *analytical modeling* leverages mathematical constructs to describe system behavior using performance for resource provisioning [67, 112], parallelism calculation and adaption [158, 130, 173] applying *monitoring approaches*, *heuristic or rule-based* methods, *queuing and control theory*. These methods offer precision for well-understood scenarios albeit at the risk of *oversimplification* in complex and dynamic distributed environments, leading to *monitoring overhead*, *inaccurate estimates*, and often limited to specific optimizations. *Benchmarking methods* offer a versatile platform for system behavior exploration by simulating real-world scenarios for understanding the performance of DSP systems [42, 111, 204, 27, 125]. It involves the execution of standardized workloads (data streams and queries) and resource configurations to assess system performance empirically under a set of pre-defined conditions [197, 79]. While benchmarking offers concrete data on system performance, translating these insights into actionable strategies requires careful consideration of the benchmarks' relevance to real-world operational scenarios. Existing benchmarking systems lack in terms of *expressiveness* to tackle the modern demand of parallel and distributed stream processing (PDSP) such as parallelism, distributed and heterogeneous environment, and support for integration of ML methods.

...to understand the performance.

Conversely, *Learned or machine learning (ML)* models leverage historical data to train predictive algorithms crucial for optimization and adaptive operations such as operator placement, parallelism tuning, and resource utilization. However, the dynamic and heterogeneous nature of DSP systems means that these models must continuously evolve, often necessitating extensive datasets for training to achieve prediction accuracy. Recently, the adaption of ML-based approaches has significantly improved in DSP system. We categorize these methods for PDSP into *(i)* online and *(ii)* offline methods. *Online methods* [54, 200, 140, 201] are vital for real-time adaptation to changing data distributions and workloads, eliminating the need for offline retrain-

Learned methods for...

...performance prediction and optimization.

ing. This adaptability is especially beneficial in environments with fluctuating data characteristics, necessitating adaptive strategies for efficient processing. Conversely, *offline learning* [4, 5, 8, 107, 244] methods for performance modeling of DSP system involve training models on a pre-defined dataset before deploying them into the system. Unlike online learning methods, where models learn and adapt incrementally as data arrives, offline learning processes the entire dataset in a batch mode to build a model. This model is then applied to the stream processing tasks without further modification during its operational phase. Offline learning is beneficial for establishing strong foundational models that can handle expected data patterns and distributions.

In this dissertation, our focus is on performance modeling in the form of *benchmarking* and *ML-based performance prediction* of DSP workloads, used for optimization decisions for parallel DSP systems like initial parallelism tuning (cf. Section 1.2). In the following section, we present insight into existing benchmarking systems, and possible pitfalls and research gaps in Section 2.2.1. We provide a brief overview of ML concepts, analyze both non-learned and learned methods, and discuss potential drawbacks in existing methods in Section 2.2.2, which leads to our research challenges.

2.2.1 *Benchmarking Methods*

Our investigation suggests that a comprehensive study of placing parallel data flow graphs across heterogeneous resources could provide valuable insights into the interactions between different operators and hardware configurations. We categorize existing benchmarking systems [125, 27, 132] for DSP into three groups: *(i)* DSP systems, *(ii)* TPC, and *(iii)* ML benchmarks.

Benchmarking systems focus...

...on sequential and batch processing.

Benchmarking Systems for DSP Systems. Despite numerous benchmarks for database management systems, standardized and systematic benchmarks specifically designed for stream processing architectures are scarce. Our review of existing systems [123, 198, 159, 125, 151, 110, 134, 34, 35] reveals significant gaps, particularly in addressing the nuances of DSP. The LRB [20] and similar efforts like the YSB [64, 65, 102] and BigDataBench [250], have expanded the scope of benchmarks but often remain focused on batch processing rather than real-time streaming. Emerging micro-benchmarks such as HiBench [121], StreamBench [251], RIoTBench [214] and OSPBench [238] bring advancements in streaming benchmarks but still fall short in adequately testing scalability and handling real-time streaming requirements. These benchmarks typically do not address essential aspects of DSP systems like parallelism, hardware diversity, and variable workloads comprehensively. Recent benchmarking systems such as DSPBench [42] and SPBench [91] address some aspects of parallelism but do not provide a holistic view of parallel stream processing. They often neglect critical elements such as the degree of parallelism and data partitioning strategies and typically rely on homogeneous hardware setups, which do not reflect real-world DSP environments [267].

TPC Benchmarks. TPC [189] has developed several benchmarks to evaluate various aspects of computing systems, primarily focusing on transaction processing systems (TPC-C, TPC-E) [61], data warehousing (TPC-H, TPC-DS) [38], [178], and other specific domains like virtualization (TPCx-V) [40], hyper-converged infrastructures (TPCx-HCI) [224], and big data (TPCx-BB) [48]. Among these TPC benchmarks, TPCx-IoT [190] is the most relevant to DSP as it addresses scenarios involving continuous data ingestion and real-time analytics, which are core to stream processing.

TPC benchmarks fall short in...

While the TPC benchmarks have contributed to standardizing the evaluation of transactional and analytical processing systems, there remains a distinct gap in benchmarking systems designed specifically for PDSP due to fundamental differences in system architecture and operational goals. TPC benchmarks provide extensive coverage of database and batch processing scenarios, which are more oriented towards singular query execution on static or slowly evolving data sets. These benchmarks fall short in addressing the unique requirements of DSP systems, such as real-time data processing, continuous ingestion, and immediate response to dynamic changes in data streams. In addition, none of the existing TPC benchmarks are designed to assess performance under these conditions; instead, they focus on batch or transactional processing where data latency and continuous data flow are less critical. They do not adequately test data streaming partitioning, which requires parallelism for dynamically scaling systems in distributed environments.

...addressing dynamic changes.

ML Benchmarks. In the context of benchmarking of learned component of DSP such as performance prediction, existing benchmarks like DeepBench [28], MLPerf [195], Fathom [3], and CleanML [154] have laid significant groundwork. These benchmarks are predominantly tailored to assess specific aspects of ML systems, from the underlying hardware's computational abilities to the effectiveness of algorithms on static datasets.

ML benchmarks lack in...

While existing ML benchmarks [11, 34] are valuable in evaluating the capabilities and performance of various systems, a significant gap exists in terms of incorporating these findings into predictive performance models. For instance, DeepBench and MLPerf primarily focus on predefined tasks for benchmarking the raw performance of hardware and machine learning frameworks but do not extend to predict future system performance based on evolving workloads or system changes. They do not provide insights into system performance under fluctuating workloads or infrastructure changes, nor do they

...predictive performance models.

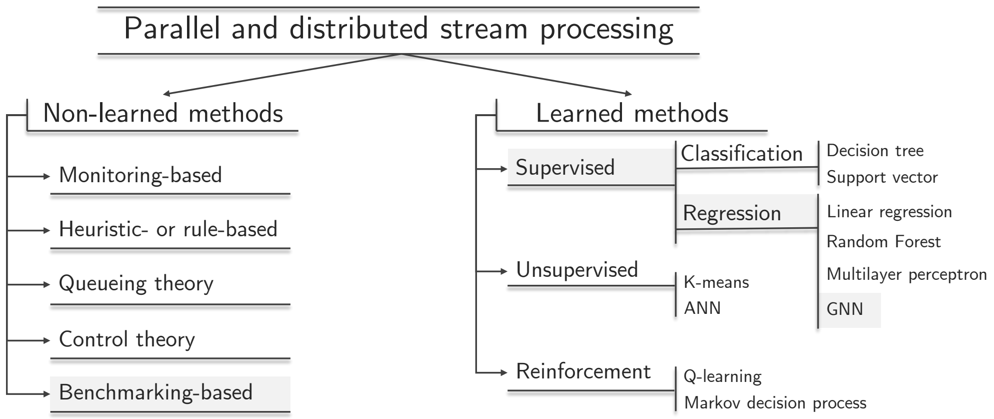


Figure 8: Overview and categorization of methods proposed for various performance and optimization tasks in DSP including resource provisioning, elasticity as well as auto parallelization. The gray-shaded area indicates the direction of our proposed performance modeling methods.

incorporate real-time data streams crucial for continuous decision-making and predictive analytics in dynamic environments.

2.2.2 *Performance Prediction Methods*

Need for predictive performance methods

The performance characteristics of individual resources in the cloud environment used in DSP systems are highly variable, necessitating a deep understanding of each resource employed [122]. Traditionally, performance evaluation models are either monitoring-based [191, 187, 207, 145] or analytically [130, 209, 53] tailored to meet the specific requirements of the executing hardware or resources. For example, models discussed in [163] are crafted to gauge the effectiveness of different operator placement strategies under ambiguous framework conditions. Therefore, there is a critical need for prediction models to predict performance and optimize the operator execution mechanisms. Our analysis of existing literature [72, 198, 66, 170, 172, 239, 180] distinguishes between *(i)* non-learned (Non-ML) and *(ii)* learned (ML) approaches in performance modeling and resource provisioning for DSP.

Non-Learned Approaches

Modern DSP systems such as Apache Flink [52], Heron [144], Storm [124] and Spark [205] incorporate optimization, e.g., parallelism degree, as a manual knob to initially tune the parallelism, enabling initial optimization based on specific workload requirements and desired performance levels. Despite the ease of tuning, manual methods are notoriously challenging to optimize reliably, as it is highly sensitive to fluctuations in DSP workload characteristics and has a high monitoring overhead.

In response to the challenges of manual tuning, significant research has been focused on deploying *online* analytical methods to monitor system performance [191, 187] for parallelism and resource provisioning for scaling decisions. For instance, various monitoring-based heuristic methods [131, 83, 260, 152, 1, 103] employ real-time metrics—like incoming and processing rates, along with CPU usage—to determine optimal parallelism degree. Similarly, mathematical optimization methods [130, 209, 53] such as game theory [135] and queuing-theory [157, 174, 213], have been developed to refine parallelism tuning processes. These methods aim to balance workload distribution and resource utilization effectively.

Existing DSP systems...

...are based on manual tuning.

Additionally, some researchers have explored analytical models for estimating necessary resources [112, 67] and adjusting parallelism [213, 140] accordingly. These models strive to automate the tuning process to some extent, offering a semi-automatic approach to determining the most efficient parallelism degree for given workloads. However, a critical limitation of these Non- ML approaches is their dependency on query runtime observation data for tuning decisions. This reliance renders them less effective for establishing an optimal parallelism level at the outset of system deployment. Consequently, these methods may lead to less than optimal initial configurations and incur significant monitoring overhead as they iteratively adjust toward ideal settings.

Continuous performance...

By investigating these non-ML strategies, we gain valuable insights into their strengths and limitations in the context of DSP systems. This understanding lays the groundwork for exploring how ML-based approaches might address some of these identified challenges, potentially offering more reliable and efficient solutions for optimizations, such as placement or parallelism tuning in dynamic and complex streaming environments.

...monitoring overhead.

Learned Approaches

Recently, there has been an interest in developing more learned or ML-based approaches for optimal operator placement [107], parallelism tuning [5, 4, 8, 244] as well as resource provisioning and elasticity decisions [54, 217, 174, 201, 200, 108] in DSP systems [192, 99, 244]. We categorize these approaches into *(i)* online and *(ii)* offline methods to understand the concepts related to ML, followed by existing research work that has been proposed to solve optimization and performance modeling tasks.

ML for...

...accurate performance modeling.

Offline methods for performance modeling involve training models on a predefined dataset before deploying them into the system. Normally, *supervised* learning is well known for offline ML, where labeled historical data is used to predict future outcomes. For instance, predicting the performance of the system to make scaling decisions. The *supervised* methods are further categorized into *classification* methods, categorizing data into predefined classes and helpful in identifying types of transactions, user actions, etc. For instance, DSP system for an online banking platform employs a classification model to identify and flag potentially fraudulent transactions [86]. The model has been trained on historical data labeled as “fraudulent" or “legitimate" and can now classify incoming transaction events in real-time, thus preventing fraud. On the other hand, regression methods play a crucial role in predicting continuous values. This is particularly essential in estimating metrics like end-to-end latency, CPU load, processing time, throughput, and more, providing a comprehensive understanding of the system’s performance. For instance, an e-commerce company uses regression models [82, 153] to predict the server load (number of requests) during peak sales or significant promotional events. For this, the regression model can use historical data of traffic, sales volume, and server metrics to predict the expected load.

In this thesis, we are focusing on *multi-regression* problem using *supervised learning* to develop methods for performance modeling. It involves predicting multiple dependent variables or outcomes, such as the performance of query execution (latency and throughput) using a set of independent input variables such as workload (query and data) and resource configuration in the context of PDSP. This type of regression is helpful to identify the correlation between different input variables, e.g., influence of multiple parallel instances of operators in the parallel data flow, and predict performance. For instance, the predicted performance from ML-models can be used to determine parallelism with minimum latency and maximum throughput to improve the overall performance of DSP systems. Different methods can be applied to regression tasks in *supervised learning* in the context of performance modeling for PDSP.

Linear regression...

...for simple regression tasks.

Linear regression [13, 175] models the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data. It is a popular method for regression problems, such as predicting parallelism [101] or CPU usage [69] based on incoming data rate. For example, if historical data shows an increase in CPU load with rising data rates, a linear regression model can quantify and predict this relationship. Linear regression for neural networks involves a single linear layer that maps input features to a continuous target variable through linear combination, without hidden layers or non-linear activation functions. This approach, while simple, allows for training using *gradient descent* and *backpropagation*, effectively bridging traditional linear regression and neural network paradigms.

Decision Tree [219, 29] is a flowchart-like structure where each node represents a test on an attribute, each branch represents the outcomes, and leaf nodes represent class labels (decision taken after computing all attributes). For example, it can decide whether to scale resources up or down based on system metrics like memory usage, CPU load, and latency, navigating these metrics non-linearly to make a binary decision.

Random Forest [59, 45] is a ML method that creates multiple decision trees during training to make predictions. For classification tasks, it uses the majority vote of the trees, and for regression tasks, it calculates the average prediction of the trees. Random forests are less likely to overfit compared to single decision trees. They offer more dependable predictions by averaging the results from many trees[141]. For instance, random forest can be used to predict the time it would take to process a batch of data streams.

Multilayer Perceptrons (MLPs) [120, 21] are a kind of artificial neural network distinguished by their architecture, which comprises at least three layers of nodes: *input layer*, one or more *hidden layers*, and *output layer*. MLPs employ a supervised learning algorithm known as backpropagation for effective training to capture non-linear complexities and could detect subtle patterns in historical data to make these predictions. For instance, MLPs can be used to predict the performance of a DSP system based on various workload and resource characteristics.

MLP can be used...

...for performance forecasting in DSP.

Convolutional Neural Network (CNN) [14] are deep learning algorithms that can take in an input image, assign importance (learnable weights and biases) to various objects in the image, and differentiate one from the other. In the context of DSP [104, 259], CNNs might be less common for performance modeling as they are more suited for image processing. However, they could also be used for real-time pattern recognition in streams of spatial or temporal data, such as image or video sequences.

Deep Neural Networks (DNNs) [206, 172] are sophisticated artificial neural networks distinguished by having several hidden layers that lie between the input and output layers. These networks are adept at modeling intricate non-linear relationships. For instance, anomalies can be detected in a complex multi-sensor environment where data streams from different sources are combined to determine the system's health or predict failures. In recent years, there has been an increasing interest in developing more efficient and robust algorithms using DNNs [194], which can be used for predicting the performance (e.g., latency or throughput) at runtime of individual operators. These models are developed to shift from manually or analytically designing performance models to more prediction-based performance methods that can be used for execution tasks such as operator placement or operator parallelization at runtime. Existing approaches [136, 169, 223, 118, 116] propose the

DNNs suffer...

...with generalization of models.

use of DNNs to develop performance evaluation models, enabling automated learning for complex operations instead of manual creation or adaptation.

However, these approaches suffer from the general problems associated with the use of DNNs. This concerns not only *high training costs* in learning the models but also the *robustness* of the predictions of DNNs, an essential property for dynamic and time-critical requirements of DSP applications. Additionally, differentiable programming represents a recent advancement in the field of ML. Unlike DNNs, which often rely on highly parameterized black-box models, differentiable programming focuses on simpler white-box models that leverage problem structure [249]. To address the limitations of DNNs, recent research [25, 155, 249] has explored performance evaluation models that can generalize robustly to new workloads, making them suitable for DSP systems. For instance, differentiable programming [115] is used to specify the general behavior of performance prediction even in previously unobserved domains. Differentiable programming has been successfully used for learned performance models for simple operators such as filters. However, it was an open question and challenge how the generalization to different workloads in DSP, such as complex operators and query plans, can be achieved, such as a *join.* This thesis addresses that challenge.

GNNs can enable generalization

Graph Neural Network (GNN) [239] are neural networks that directly operate on the graph structure, allowing them to take the relationships and structures within the data into account. For instance, in a DSP application like network traffic management, where data streams represent different nodes or operators and their interaction are edges, a GNN could predict the impact of an operator's performance on the overall system to optimize routing and resource allocation. Addressing the challenges posed by DNNs and differential programming requires methods, especially in achieving model generalization. Existing research within the database systems [113, 114, 106] has demonstrated the feasibility of models to generalize across various scenarios. Nevertheless, this feasibility often comes at the cost of a substantial *amount of training data* and enormous *training effort and time.* For example, established models in this field necessitate training on datasets comprising over $200k$ queries and spanning 15 distinct databases to attain a level of generalizability across previously unseen databases [107, 6, 5, 114].

In contrast to databases, DSP systems are characterized by dynamic workloads and the unpredictable nature of data stream characteristics. As such, DSP systems face unique challenges in preparing for the variety of potential workloads. The ability to support a diverse array of workloads while simultaneously minimizing the effort required to train models on unseen data has become paramount. Achieving this balance is crucial for ensuring the reliable and predictable operation of parallel operators within DSP systems. By reducing the training overhead and enhancing the model's ability to adapt for new scenarios, can significantly improve the efficiency and adaptability of DSP systems [87, 150], making them better equipped to handle real-time data processing. In this context, GNN can play an essential role in addressing accuracy and generalizability in performance prediction for DSP systems which can further used for determining optimal operator placement [107, 106] and parallelism [6, 8].

Online methods are vital for real-time adaptation to changing data distributions and workloads, eliminating the need for offline retraining [54, 200, 140, 201, 72, 112, 244, 180]. This adaptability is especially beneficial in environments with fluctuating data characteristics, necessitating adaptive strategies for efficient processing [95, 194, 254]. There are two online ML methods that are commonly used for various optimization in DSP systems: *Unsupervised* and *Reinforcement.*

Unsupervised models are ideal...

Unsupervised learning focuses on discovering patterns without pre-labeled data, making it ideal for understanding complex system behaviors [12], optimizing configurations [84, 183], and identifying anomalies [243] within PDSP. It is beneficial for tasks like operator placement, parallelism tuning, and anomaly detection without explicit historical examples. For instance, unsupervised learning [84, 183] is applied to collect performance and system metrics during query execution, such as including each operator's throughput, latency, and the volume of data exchanged with other operators to identify group operators that should be placed close together to minimize data transfer latency and maximize throughput in DSP systems.

... for scenarios with no labeled data.

Similarly, *Reinforcement* learning (RL) is widely utilized in DSP to enhance decision-making through a trial-and-error method, where the system learns by interacting with the environment and receiving rewards for favorable outcomes. It proves particularly effective in dynamic and complex environments of DSP where optimal decisions can change over time [244]. For instance, Deep Q-Networks (DQN) or Proximal Policy Optimization (PPO) algorithms [200, 117, 201, 117] are used to train the RL agent which learns the best scaling policy by continuously interacting with the system simulation or directly with the live system in a controlled manner. Additionally, some studies [264, 200] in DSP have investigated stream partitioning to predict and manage parallelism using RL models. Another significant methods [108, 54, 63, 62] involve enhancing DSP systems' adaptability by dynamically allocating or deallocating resources to match fluctuating workload. Recent advancements also show a growing trend toward automating parallelism adjustment [200, 201, 80]. Despite the performance gains achieved through ML, these approaches often rely on black-box models, which can obscure the reasoning behind specific decisions. Moreover, they tend to be trained on workload-specific features, limiting their ability to generalize across different workloads and various resource configurations.

Reinforcement learning...

...ideal for dynamic adaptation.

For all the above learned approaches, traditional ML techniques typically operate under the assumption that the training and testing data come from the same domain, sharing similar input feature spaces and data distribution characteristics. This assumption holds in scenarios where a large corpus of labeled data is readily available, ensuring the models are well-tuned to the specific task within a narrow domain.

However, many real-world ML applications such as natural language processing [199, 248], image classification [211] and DSP [269] are not ideal to have all input scenario to be available often, acquiring sufficient training data that is representative of the operational settings of the model can be expensive or practically infeasible. However, many real-world ML applications, such as natural language processing [199, 248], image classification [211], and DSP [269], often face the challenge of not having all input scenarios readily available. Acquiring sufficient training data that accurately represents the operational settings of the model can be expensive or practically infeasible.

Transfer Learning

Transfer learning can...

...assist in transferring knowledge...

...ideal for DSP applications.

Transfer learning [255, 186, 275] is a ML technique which allows ML models to be trained on one task and transfer knowledge to infer on another task. This technique allows extending beyond the conventional training paradigms. It leverages existing data from related but distinct domains to enhance the *generalization* capabilities of ML models. By doing so, transfer learning addresses the critical challenges posed by the limitations of *data availability* and the expensive cost associated with *data collection* as well as *iteratively retraining* in new workloads or domains. In essence, transfer learning allows for the performance optimization of a learner by transferring knowledge from a related source domain to a target domain where data may be scarce or lacking. For instance, the task of sentiment analysis in product reviews data stream. If ample labeled data is available for the input data stream of digital movie reviews, traditional supervised or unsupervised ML models can yield robust predictive models for this specific domain. However, when the same model is applied to another target input data stream or book reviews, the performance of these models might be degraded due to variations in *domain-specific or workload-specific* data characteristics. Despite these differences, the underlying structure of the data—textual reviews expressing opinions—remains consistent. *Transfer learning* can exploit these common or similar attributes, enhancing the model's ability to generalize from movie to book review data streams by adapting the learned features and patterns to the new domain.

Therefore, this flexibility makes transfer learning particularly appealing in the era of big data or stream processing, where models can learn from a broader context than what is available in the target domain alone. The success of transfer learning has been demonstrated across a wide array of applications, from text sentiment classification [199, 248, 245] to image classification [211, 274, 143, 74], users activity or trend recognition [105], and multi-language text classification [273].

In the context of DSP, transfer learning can significantly enhance the performance modeling and prediction tasks. As shown in this work, a performance model can predict accurate costs (e.g., latency and throughput) and generalize across workloads and resources that the model did not see during training. Recent work in the database has adopted transfer learning for query optimization and cardinality estimation [107, 114]. However, the adoption of transfer learning in the field of DSP is new, and we are the first ones who have proposed the performance model for DSP [8, 5, 6].

2.3 Summary

In this chapter, we introduced foundational concepts of DSP and reviewed existing research to identify key research gaps (cf. Section 1.1) and to better contextualize the contributions (cf. Section 1.2) that this thesis addresses. Our exploration of the current benchmarking system for performance modeling reveals significant research gaps- lack of *expressiveness* for parallel dataflows and heterogeneous environments and incorporating learned or ML models into DSP systems. Current research mainly focuses on performance modeling of sequential dataflows with homogeneous resources, while modern DSP systems require parallel processing in heterogeneous environments. Furthermore, existing DSP systems extensively rely on manual tuning and continuous performance monitoring for optimization tasks such as operator placement and parallelism. While performance modeling using ML methods like DNNs require *extensive data* and *high training efforts*, struggling to *generalize* for new workloads. Online learning methods, though effective, are blackbox, making interpretation difficult and requiring multiple adjustments for optimal provisioning. This underscores the need for performance modeling methods - *(i)* benchmarking system for DSP system to understand the correlation among varying workloads (parallel dataflows and data streams), and resource characteristics and *(ii)* ML-based performance prediction models to provide accurate performance predictions and generalize to unseen workloads and resource configurations to reduce the need for iterative training and determining efficient optimization, e.g., operator placement and parallelization.

3

Scenario and Conceptual Architecture

This chapter outlines parallel and distributed stream processing (PDSP) scenario and the overall architecture of performance modeling methods proposed in this thesis. Section 3.1 presents the scenario employed throughout this thesis and the underlying research challenges that are addressed. Section 3.2 provides a comprehensive description of the architecture, incorporating formal definitions of the concepts that are fundamental to the contributions of this thesis. This structure sets the stage for detailed discussions on performance modeling methods in subsequent sections.

3.1 Scenario Description

DSP applications scenario

Distributed Stream Processing (DSP) systems have become pivotal in managing large-scale, real-time data across cloud resources in various sectors. By enabling parallel processing of streaming data, these systems provide immediate benefits in decision-making and responsiveness, which are crucial to the success of many industries. In this thesis, we have considered different valuable application scenarios where real-time data analysis is crucial for decision-making such as ① *click stream analytics*, ② *social media analytics*, ③ *healthcare monitoring*, ④ *smart city* and ⑤ *financial sectors* as presented in Figure 9. We have considered different DSP applications and corresponding real-world scenarios to show the applicability of our concepts and proposed methods to tackle the identified challenges (cf. Section 1.1) of *RC1*: understanding the performance and *RC2*: performance prediction for optimization. We briefly explain each application's possible scenarios, underlying assumptions, and possible infrastructure used in the following.

① *Click Stream Analytics*

DSP can be used to handle massive volumes of data generated from millions of users streaming videos concurrently. The analytics derived from this data are crucial for optimizing streaming quality and enhancing user experience. By analyzing data points such as viewer preferences, buffering rates, and

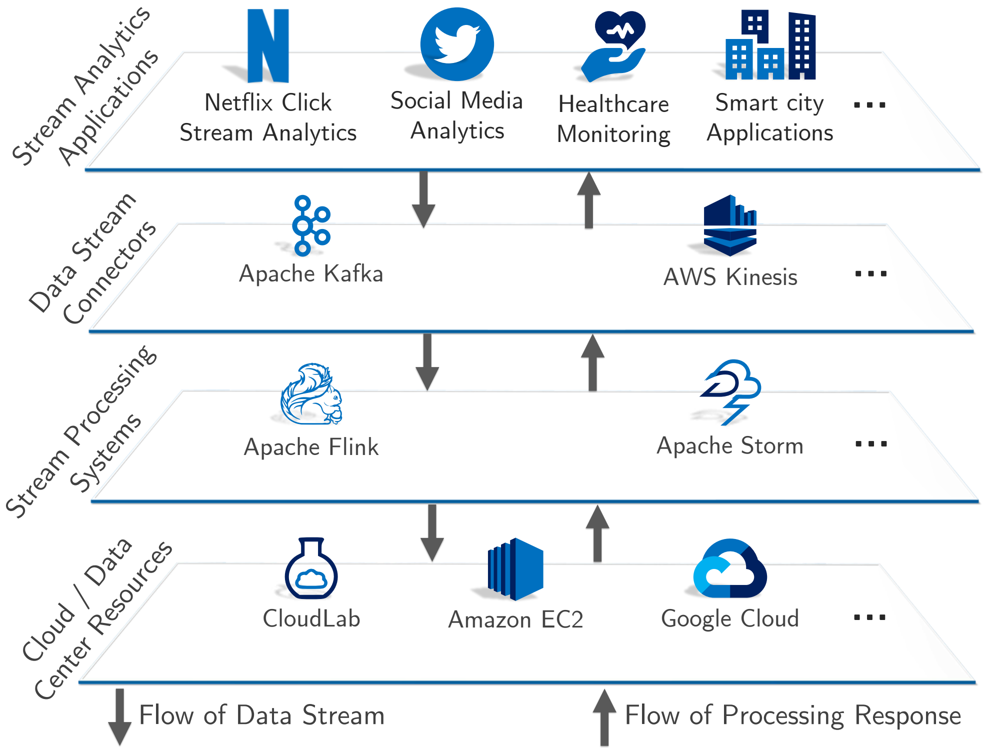


Figure 9: PDSP applications scenarios in the cloud resources to process a massive amount of streaming data: ① *Netflix* click stream analytics for enhancing user experience and personalized content recommendation ② *Twitter* social media analytics for understanding user engagement and targeted advertisement, ③ Healthcare monitoring to detect anomalies and potential health issues and immediate responses, ④ Smart city applications to manage and optimize urban services such as controlling traffic lights and reduce congestion, and ⑤ Financial fraud detection for anomalies and fraud transaction pattern detection for financial trading.

Click analytics for viewer interactions

viewing times, video streaming application providers such as *Netflix* [33] can dynamically adjust video quality to match different internet speeds and device capabilities, ensuring seamless service delivery. Additionally, this data informs content recommendation algorithms to tailor suggestions to individual user tastes and thus increase viewer engagement and satisfaction. For instance, a prime example of stream processing in Netflix is the use of its data pipeline for real-time analysis of viewer interactions. Netflix uses Flink to process over $1.3TB$ of data in its daily tasks, helping to optimize streaming quality by adjusting to bandwidth fluctuations and device capabilities. For instance, during the onset of the COVID-19 pandemic, Netflix experienced unprecedented spikes in viewer numbers (26 million new subscribers) [10] as lockdowns were implemented worldwide. Their stream processing capabilities allowed them to dynamically manage this surge and maintain smooth streaming experiences, even implementing bit rate reductions in Europe to lessen the strain on internet service providers.

② *Social Media Analytics*

Similarly, DSP can analyze social media click stream data in real-time. Big giants for social media click stream producers such as *Twitter (now known as X)* ($100K$ tweets per minute) [73] or *Facebook* ($9GB$ streams per second) [139] rely on DSP for real-time analysis. For instance, it can involve understanding how users interact with tweets, including clicks on links, retweets, likes, and replies. Such analytics are vital for understanding user engagement and the spread of information across the network. Processing these data streams, such as various Tweets streams, can deliver more personalized content to users, enhance ad targeting, and gain valuable insights into trending topics and user behavior patterns. This real-time analysis improves user experience and boosts advertising revenue through more effective campaign implementations. Consider the case of Twitter, whose real-time analytics engine [235, 124] processes massive volumes of tweets and interactions to track trends and user engagement. During significant global events, such as elections or large-scale protests, Twitter's analytics are instrumental in identifying the spread of information and misinformation, enabling them to promptly address fake news or harmful content [271]. This not only maintains the platform's integrity but also enhances its usefulness as a real-time news and social networking service, thereby significantly impacting the user experience and the platform's advertising revenue. In this scenario, every user's device that streams video content, such as smartphones, tablets, and smart TVs, generates data about the content being watched, its duration, play and pause actions, and the quality of the stream.

Personalized recommendation using DSP

③ *Healthcare Monitoring*

In the healthcare sector, DSP is employed to monitor patient data continuously, including real-time analysis of data from medical devices such as heart rate monitors, respiratory sensors, and wearable health trackers. Stream processing enables healthcare providers to detect anomalies and potential health issues as they occur, facilitating immediate medical response. For instance, sudden changes in a patient's heart rate can trigger instant alerts to medical staff, potentially saving lives through timely intervention [247, 212]. This capability is particularly beneficial in remote patient monitoring, where immediate responses to data can dictate health outcomes. For instance, stream processing systems integrated with IoT devices could continuously monitor oxygen levels and heart rates in COVID-19 patients, enabling immediate medical intervention when anomalies are detected. This capability is vital in in-

Detect anomalies and health issues

tensive care units where timely information could mean the difference in life-threatening situations.

④ *Smart City*

Traffic monitoring and control

Smart city applications where DSP are implemented intensively to manage and optimize urban services [179, 233] such as traffic management through real-time analysis of data from traffic cameras and sensors to control traffic lights and reduce congestion [16]. Additionally, DSP can support public safety by monitoring surveillance footage for unusual activities and triggering alerts to law enforcement when necessary. Similarly, waste management [177] is another area where data from sensors in waste bins can be analyzed to optimize collection routes and schedules, thus improving efficiency and reducing operational costs. For instance, traffic management systems in cities like Singapore [137, 60], where real-time data from cameras and sensors is used to manage traffic flow. During the pandemic, such systems were adapted to monitor and control the movement of people to ensure that social distancing norms were being followed in public spaces. Similarly, cities have enhanced environmental monitoring to quickly detect and address pollution hotspots, contributing to healthier urban environments.

⑤ *Financial Fraud Detection*

Fraud detection using DSP

Financial fraud detection is a critical area where DSP plays a pivotal role, particularly as financial transactions continue to grow in volume and complexity with the expansion of digital banking and e-commerce [168]. For instance, *Infront Financial Technology* [85] handles approximately 24 billion events daily, equating to about $300,000$ events per second, to promptly execute pattern detection queries in financial trading. Similarly, *PayPal* [220] employs machine learning models that run on distributed systems to analyze transactions across its platform. The system checks each transaction against the account holder's usual patterns and flags transactions that deviate significantly. For instance, it looks at the transaction size, the device used, and the transaction's location. This analysis happens in real-time, leveraging the power of distributed stream processing to maintain system performance even as it scales to handle millions of transactions daily.

Summary of Scenarios

In the above application scenarios, the stream processing encompasses a suite of analytics applications that deliver services based on real-time data analysis, such as video streaming analytics, as shown in Figure 9. Each application harnesses *data stream connectors* such as *Apache Kafka* or *AWS Kinesis* to ingest voluminous data streams generated from diverse sources. These sources might include user preferences for Netflix video content, traffic flow information from urban sensors, or transactional data from financial institutions. In our analysis, we employ these *data stream connectors* to funnel preexisting data from these varied sources for stream processing tasks tailored to each application domain. Moreover, specific queries corresponding to each service are employed to process the incoming data streams. These queries dictate the configuration of the operator flow graph, which is subsequently deployed on the available cloud resources to process the data streams.

Bringing all scenarios together in a holistic architecture...

Furthermore, we consider cloud infrastructure, which plays an essential role, comprising a variety of hardware resources, including CPUs and GPUs with diverse configurations tailored for stream processing as depicted in Figure 9. Such infrastructure ranges from open-source testbeds like *CloudLab* [76], which offers a controlled experimental environment, to industrial-scale cloud services like *Amazon EC2* [129], *Google Cloud* [252], renowned for their robustness and scalability in handling stream processing tasks. Stream processing systems like *Apache Flink* [52] and *Apache Storm* [124] can be deployed on cloud resources to process and analyze data streams.

...proposed in this thesis.

Within the diverse scenario of different applications, the parallel instances of stream processing operators emerge as a strategic solution to manage high-volume workloads, scaling decisions, and meet QoS requirements. However, the diversity in these application requirements poses the significant challenge of parallelism tuning—the process of determining the optimal number of parallel operator instances based on varying workload characteristics. Effective parallelism tuning is critical, as it directly impacts the ability of a DSP system to process data streams efficiently and maintain the desired QoS levels. For instance, a financial trading platform must process fluctuating volumes of transactional data in real-time. During peak trading hours, the workload (volume of data) can spike dramatically, necessitating rapid parallelism level to maintain real-time processing capabilities without sacrificing accuracy and QoS, i.e., ensure latency remains low without compromising on throughput. The challenge lies in preemptively understanding how these adjustments will affect performance before they are implemented, as any misstep could lead to costly downtime or erroneous transactions. Conversely, a healthcare monitoring system may require different parallelism settings to process disparate data types, such as continuous heart rate monitoring versus sporadic alert notifications from medical sensors.

Accurate performance modeling is essential...

To effectively manage operator parallelism, it is crucial to precisely understand the resulting performance before implementing any changes. However, these challenges are further compounded by the unpredictable nature of DSP workloads. Unlike databases, data stream characteristics—such as volume, velocity, and variety—can change without prior indication, making it difficult to anticipate the system's behavior under different conditions. As such, an alteration in the degree of parallelism necessitates careful reconfiguring of DSP queries, which may involve the expensive reallocation of resources and division of operational states among the newly scaled set of operators in cloud resources. To overcome these challenges, we introduce PDSP-BENCH, a benchmarking tool that evaluates performance across varying workloads and degrees of parallelism within DSP systems. Alongside, ZEROTUNE offers a predictive performance cost model, estimating metrics such as latency and throughput even for previously unseen workloads and resource configurations. This predictive capability is critical for initial parallelism tuning, ensuring a cost-effective approach to stream processing in cloud-based environments tailored to the real-time demands of these applications.

...as provided by this thesis.

3.2 Conceptual Architecture

In this section, we explain the overall architecture of the proposed performance modeling and parallelism tuning methods in this thesis. Section 3.2.1 outlines a comprehensive system model, providing definitions relevant to the contributions discussed throughout this thesis. Section 3.2.2 elaborates on the overall architecture and the interactions between its components.

3.2.1 *Performance and Parallelism Tuning Model*

The proposed performance modeling methods support data sources or producers (S_O) such as IoT devices and applications that generate data streams (D), as well as data consumers or sinks (S_I) that express interest in the produced events by formulating a set of DSP queries (Q). Each query ($q \in Q$) comprises a set of operators ($\omega \in \Omega$) that can be represented as a directed acyclic graph G = ($S_O \cup \Omega \cup S_I$, D). Here S_O, Ω, and S_I are represented by the set of vertices ($v \in V$), and the edges ($e \in E$) in the graph (G) indicate the flow of the data stream between operators. Data sources (S_O) have no upstream operators, and data sinks or consumers (S_I) have no downstream operators. These operators can be deployed on a set of n $\in \mathbb{N}$ processing resources $R = \{R_1, ..., R_{max_r}\}$ based on the QoS requirements of data sinks (S_I).

In the following, we describe the system model of the proposed system that enables conceptual understanding and analysis of the described *transferable features* for data, query, parallelization, and resources, as well as facilitate understanding of the cost metrics.

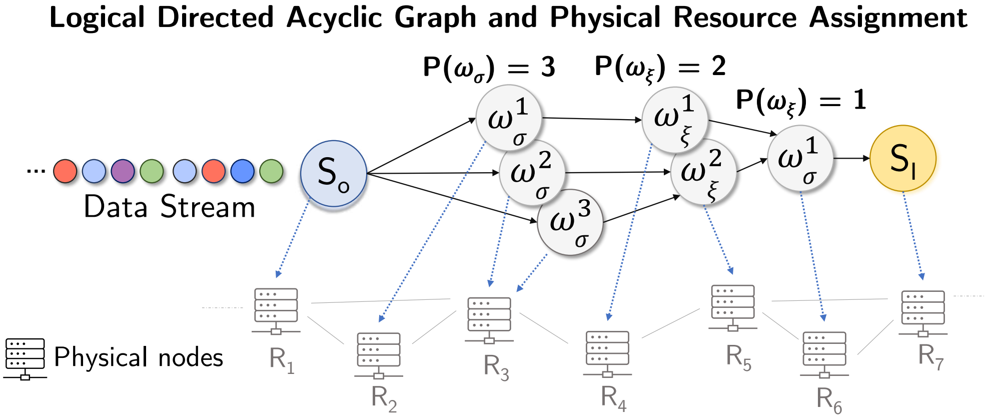


Figure 10: Data and query model for logical directed acyclic graph and physical resource assignment of operator in DSP systems [9, 8].

Data and Query Model

Data Stream Model: The data stream (D) is an *unbounded* series of event tuples $t_i \in D$, where $t_i =< t_1, t_2, \dots >$. Each tuple t = (τ, k, d) comprises three elements: a logical timestamp τ that indicates when the event occurred, an event key k that serves as a unique identifier for the event, and event data d that represents the payload associated with the event. To process the event tuples t_i in a query q, they are bounded by window (w). The window w in a stream D has two attributes *window length* (w_l), and *window slide* (w_s). In the proposed methods, we focus on the *count*, *time*, and *key-based* windows that differ in size based on count, time, and keys, respectively. Further, we consider moving the window as (1) *tumbling* and (2) *sliding window*, which are well-suited to a wide range of use cases, and well-supported by various DSP systems [52, 144, 205].

Query process data using operators

Operator and Query Model: A data stream of tuples t_i is processed using a set of operators (Ω) as illustrated at the top of Figure 10 (in circles). These operators can be used to perform key operations on the data stream and are critical for many use cases. The proposed methods (PDSP-BENCH and ZEROTUNE) supports a set of widely used standard DSP operators: filter (ω_σ), join ($\omega_\bowtie$), and aggregation operators (ω_ξ) such that $\Omega = \{\omega_\sigma, \omega_\bowtie, \omega_\xi\}$. While we limit the scope to these operators, our methods can be generally applied to other operators by extending the transferable features based on our methodology. These operators can be parallelized to form their parallel query plans. In Figure 11, we illustrate the logical parallel query plan for two-way join with different parallelism degrees. In a distributed setting, these combinations and configuration options offer a rich set of possible outcomes, which can be used to collect a large set of training data for our learning model.

Query consists of multiple operators

Parallelism for processing higher workload

Notations	**Meaning**
S_O, S_I	Set of data sources and data sinks
D	Unbounded data stream
Ω	Set of operators, $\omega \in \Omega$
D_{In}, D_{Out}	Incoming and outgoing data stream to/from operator
t	event tuple, $t = (\tau, k, d)$(timestamp, key, payload)
G	Directed acyclic graph
Q	Set of queries, $q \in Q$
W	Set of windows, $w \in W$
$h(t)$	Hash value of t
P	Set of parallelism degrees, $p \in P$
ω_i^{ins}	Number of parallel instances (ins) of operators ω_i where $ins \in \{1, 2, ..., max_P\}$
R	Set of physical resources, $r \in R$
n_{core}	Number of available cores on physical node
max_P	Maximum possible parallelization $max_P = n_{core}$
L, T	Latency and throughput metrics
U_{CPU}, U_{mem}, U_{nw}	Resource utilization such as CPU, memory, network usage
Sel_ω	Selectivity of operators, $\omega \in \Omega$
L_ω	Processing latency of operators, $\omega \in \Omega$

Table 2: Terminologies and their meaning.

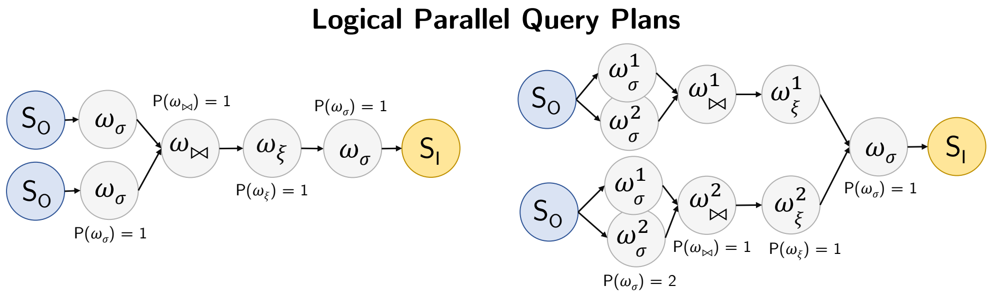


Figure 11: Example of how logical parallel query plans are represented in the proposed performance modeling methods for join and 2-way join queries.

Parallelization Model

As mentioned before, the proposed methods support *data parallelization* [198, 119] where multiple instances of an operator (ω_i) are created, and data streams are partitioned to those instances ω_i^{ins}, where $ins \in \{1, 2, \ldots, max_P\}$ refers to each parallel core of resource r that executes operators concurrently. The parallel instances ω_i^{ins} could be placed either on the same or different physical resources (R) as shown in Figure 10. The maximum available parallelization ($\max_p$) of all operators $\omega \in \Omega$ will be restricted by the available characteristics of the resource, i.e., the total number of CPU cores (n_{core}) of all the physical resources (max_r), $\max_P \leq \mathrm{n}_{core}$.

Different parallelization schemes for operators

Parallelization Schemes: The proposed methods support several partitioning schemes [52, 50, 232] that determine how a data stream is partitioned across the parallel instances of an operator ω_i^{ins}. For instance, the filter operator employs *forward partitioning* to forward data to the next operator instance in the pipeline without any modification with the same number of operator instances. If the data is unevenly distributed, then the filter operator uses *rebalance partitioning*, a round-robin scheme to redistribute data across all parallel instances of an operator evenly. Similarly, window aggregation and join use *hash partitioning* based on a specific key. In addition, the parallelism of operators influences the partitioning scheme selection. For instance, when both upstream and downstream operators have the same parallelism degree, the *forward scheme* forwards the data stream to the downstream operator, whereas *rebalance* distributes the datastream tuples in a round-robin fashion to the downstream operator instances. Finally, the *hashing scheme* allocates tuples using a hash key when the downstream operator has a higher degree of parallelism compared to the upstream operator. In the following, we define the schemes formally.

Partitioning defines data distribution...

...between operator and its instances.

Let ω_i and ω_j represent the upstream and downstream operators, respectively, and $P(\omega_i)$ and $P(\omega_j)$ denote the parallelism degree of each operator ω_i and ω_j, respectively. Let t be a tuple in the datastream D and $h(t)$ be the hash value of t based on a specific key. Then, the partitioning of the data stream based on the operators and their parallelization will be as follows:

Partitioning depends on operator type

Definition 3. *Forward Partitioning:* Given a data stream *D*, set of parallelism degree P, if $P(\omega_i) = P(\omega_j)$ then forward partitioning assigns each tuple *t* in *D* to a single instance ω_j^{ins} in a sequential manner, such that the i^{th} tuple of t is assigned to the $(i \textit{ mod } n)^{th}$ instance of operator, where *n* is the number of instances in P.

Filter uses rebalance strategy

Example: Filter use forward partitioning where each parallel instance of the upstream operator sends its data directly to the corresponding parallel instance of the downstream operator based on their subtask indices. Thus, in this strategy, no data shuffling or distribution is required.

Definition 4. *Rebalance Partitioning:* Given a data stream *D* and set of parallelism degree P, if $P(\omega_i) \neq P(\omega_j)$, rebalance partitioning assigns each tuple *t* in *D* to a single instance ω_j^{ins} such that the number of elements assigned to each instance is approximately equal.

Example: Filter use rebalance partitioning [52, 50, 232] for non-congruent parallelism degrees when the workload needs to be evenly distributed across all parallel instances of the downstream operator, regardless of the key.

Definition 5. *Hash Partitioning:* If $P(\omega_i) < P(\omega_j)$, then the tuple t is distributed based on the hash value of t as follows: t is hashed to a specific value $h(t)$, and then the tuple is sent to the parallel instance of the downstream operator whose index is the remainder of $h(t)$ divided by $P(\omega_j)$. When there is a key-based relationship between records, such as grouping, joining, or aggregation operations, hash partitioning is applied.

Example: Window Aggregation. Given a data stream *D*, a set of parallel degrees (*P*), a key function *k*, and a window function *w*, hash partitioning for window aggregation assigns tuples *t* in *D* to windows and then partitions the windows across the parallel instances in *P*. The assignment of tuples to windows is done by applying the window function *w* to each tuple *t* in *D*, resulting in a set of windows *w* = $\{w_1, w_2, ..., w_{\mathrm{m}}\}$. Then, each window w_{i} is assigned to a single instance $p \in P$ by computing *h = hash(k(w$_i$)) mod n*, where $hash$ is a hash function that maps the key to a non-negative integer, and *n* is the number of instances in *P*. The window *w$_i$* is then assigned to the instance $P_h = P_{h+1}$.

Hash partitioning for window operator

Example: Window Join. Given two data streams D_1 and D_2, a set of parallel instances P, and key functions k_1 and k_2, hash partitioning for window join assigns tuples in D_1 and D_2 to windows and then partitions the windows across the parallel instances in P. The assignment of tuples to windows is done similarly to the window aggregation case by applying the window function *w* to each tuple *t* in D_1 and D_2, resulting in sets of windows $w = \{w_1, w_2, ..., w_m\}$ and $w' = \{w'_1, w'_2, ..., w'_m\}$. Then, each window $w_i \in w$ and $w_j \in w'$ is assigned to a single instance in $p \in P$ by computing $h_1 = hash(k_1(w_i))$ mod n and $h_2 = hash(k_2(w_j))$ mod n, where *hash* is a hash function that maps the key to a non-negative integer, and *n* is the number of instances in P. The windows w_i and w_j are then assigned to the same instance if $h_1 = h_2$.

Resource Model

Different resource characteristics

The resource model of the proposed methods encompasses a broad spectrum of distributed and heterogeneous cloud, fog, and IoT/edge resources with diverse configurations, as required by several DSP applications. At present, the system focuses on *CPU-based hardware*, but our model can also capture heterogeneous hardware architectures such as *GPUs* [139, 268, 242, 193, 30, 229, 37], *FPGAs* [202, 57, 176, 277], P4 switches [126, 276, 148, 93, 94, 92, 226], given support in DSP systems. The system assumes that IoT devices or applications at the edge layer will serve as data sources and produce one or multiple data streams. These data streams will be processed on the available processing physical resources (*R*) in the fog or cloud layer by deploying parallel instances of operators, for which the proposed methods predict parallelism degrees.

Metrics for Performance Model

The aim of the proposed methods is to depict parallelism degrees for operators on physical resources such that the overall performance or cost[3] of execution on the given resources is minimal. We consider two main performance metrics widely discussed in DSP literature [221]: end-to-end latency and throughput [133, 164, 42]. While we consider these two metrics for the parallelism degrees prediction task, our model can be fine-tuned for other cost metrics by replacing the final Multilayer Perceptron (MLP) decoder with another metric.

Definition 6. *End-to-end latency (L)*: This metric quantifies the total time taken from when the first data tuple is generated at the source (S_o) to when the query result is output at the sink (S_i). It comprises the processing latency ($L_{proc}(\omega)$) of each operator ($\omega \in \Omega$) in the processing pipeline, the network latency ($L_{nw}(\omega)$) incurred during data transmission from sources to sinks, and the latencies associated with data input (L_{in}) and output (L_{out}) to external systems, such as IoT platforms. We consider network latency ($L_{nw}(\omega)$) into end-to-end latency as operators and their instances may be allocated on the same resource or distributed across different resources. Formally, end-to-end latency [8] is defined as: $L = L_{in} + \sum_{\omega}(L_{proc}(\omega) + L_{nw}(\omega)) + L_{out}$.

Definition 7. *Throughput (T)*: This metric quantifies the rate at which the DSP system processes data records, measured in the number of records processed per unit of time [164]. In contrast to existing work that measures throughput sustainably, i.e., without backpressure [133]; we consider backpressure in our measurement to account for where there is a need to increase the parallelism degrees.

Additional Metrics for Performance Model

Apart from *end-to-end latency* and *throughput*, the proposed methods ensure the accurate performance modeling of other metrics such as *selectivity* and *processing latency* of an operator to facilitate a nuanced understanding of query execution performance. Moreover, *resource utilization* [42, 111] – spanning *CPU*, *memory* and *network* metrics–is a critical aspect for parallel and

[3] Performance and cost are used interchangeably in this thesis.

distributed stream processing, which can affect performance of DSP systems for query execution. The proposed methods aim to collect and analyze resource consumption metrics to examine DSP system's behavior under varying parallelism with changing the workload configurations.

Essential to measure...

For instance, consider a DSP system handling real-time analytics for an e-commerce platform during a flash sale event. Here, the selectivity metric helps determine the efficiency of a filtering operator that sifts through millions of transaction events to identify those related to the flash sale. If the operator's selectivity is low, it indicates that the system is processing a high volume of irrelevant data, thus calling for query optimization. Similarly, consider CPU usage peaks during the flash sale for resource utilization. An unusually high CPU utilization may indicate that the system is either *under-provisioned* or that the current query parallelism degree is not optimal, leading to potential bottlenecks. Conversely, low CPU utilization might signal *over-provisioning*, thus incurring unnecessary costs.

...performance and resource usage.

These performance measures are crucial for calibrating DSP systems to achieve optimal resource efficiency while maintaining high performance. Currently, the proposed methods support these metrics for accurate performance modeling, but they are not limited to these metrics and can be easily extended for additional examination or fine-tuned for performance predictions. We briefly define these metrics in the following section.

Definition 8. Selectivity (Sel_{ω}): Selectivity of an operator $\omega \in \Omega$ is defined as the ratio of the number of output tuples D_{out} to the number of input tuple D_{in} for ω:

$$Sel_{\text{op}}(\omega) = \frac{|D_{\text{out}}(\omega)|}{|D_{\text{in}}(\omega)|}. \tag{1}$$

Definition 9. Processing Latency of Operators (L_{proc}): The Processing Latency of an operator $\omega \in \Omega$ is the time taken to process an input tuple, defined as the difference between the tuple emission time and the tuple reception time:

$$L_{\text{proc}}(\omega) = T_{\text{emit}}(\omega) - T_{\text{receive}}(\omega), \tag{2}$$

where $T_{emit}(\omega)$ and $T_{receive}(\omega)$ are the timestamps of emission and reception of the tuple by operator ω, respectively.

Definition 10. CPU Utilization (U_{CPU}): It is defined as the ratio of time the CPU is actively processing tasks related to the stream processing application compared to the total observed time. For a given set of physical resources R and a set of parallelism degrees P for operators Ω in a directed acyclic graph G, the CPU Utilization for a physical node $r \in R$ during query execution is defined as:

$$U_{\text{CPU}}(r) = \left(\frac{\sum_{p \in P} \sum_{\omega \in \Omega} T_{\text{active}}(\omega_p, r)}{T_{\text{total}}(r)} \right), \tag{3}$$

where $T_{active}(\omega_p, r)$ is the time spent by operator instance ω_p actively executing on resource r, and $T_{total}(r)$ is the total time of observation for resource r.

Definition 11. Memory Utilization (U_{mem}): Memory Utilization measures the proportion of total memory being used by the stream processing application. For a physical node $r \in R$, the memory utilization is defined as the ratio of the memory used by the operator instances to the total available memory on r during query execution:

$$U_{\text{mem}}(r) = \left(\frac{\sum_{\omega \in \Omega} M_{\text{used}}(\omega, r)}{M_{\text{total}}(r)} \right), \tag{4}$$

where $M_{used}(\omega, r)$ is the memory used by operator ω on resource r, and $M_{total}(r)$ is the total memory available on r.

Definition 12. Network Utilization (U_{nw}) For a physical node $r \in R$, the network utilization is the ratio of the data transmitted by the operator instances to/from r to the maximum network bandwidth capacity of r:

$$U_{\text{nw}}(r) = \left(\frac{\sum_{\omega \in \Omega} B_{\text{transmitted}}(\omega, r)}{B_{\text{max}}(r)} \right), \tag{5}$$

where $B_{transmitted}(\omega, r)$ is the data transmitted by operator ω to/from resource r, and $B_{max}(r)$ is the maximum bandwidth of r.

3.2.2 *Architecture and Contributions Overview*

In this section, we present a comprehensive overview and the overall conceptual architecture of the proposed performance modeling methods in this

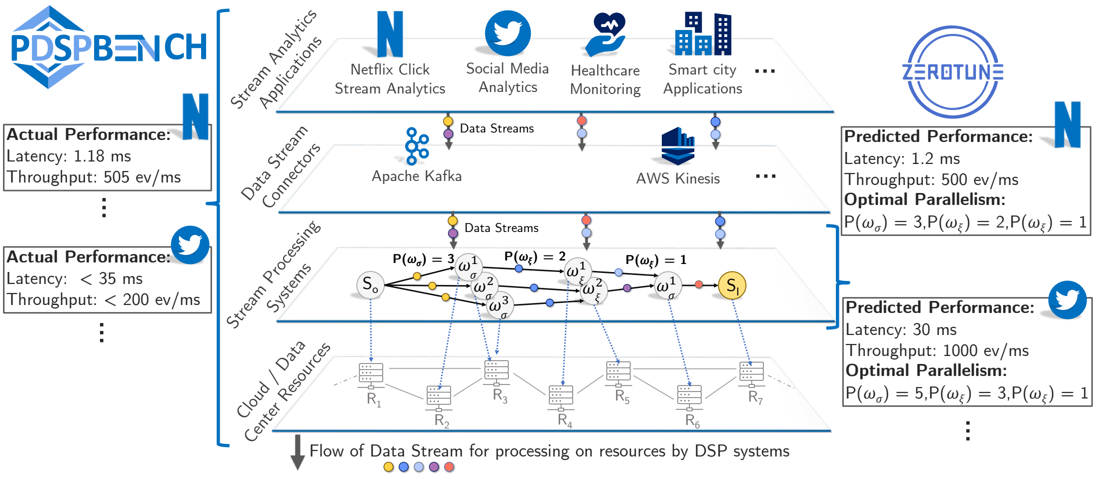


Figure 12: Overall architecture of the proposed performance modeling and parallelism tuning methods. *(i)* The system consists of different *streaming analytics applications* which are producing *data streams* continuously. *(ii) Data stream connectors* act as pipelines to forward these streams to DSP system where data processing query is represented as DAG. *(iii)* PDSP-BENCH is used to understand the actual performance of query execution on cloud/data center resources with varying workloads and generate performance data, i.e., query configurations and performance labels to be used to train machine learning models. While *(iv)* ZEROTUNE is used to predict performance without query execution and using predicted performance to determine optimal parallelism degree (e.g., minimum latency and maximum throughput) of each operator in DAG.

thesis, as shown in Figure 12. The foundational concept and preliminary methods were initially introduced in Chapter 1; here, we further elaborate on the refined architecture based on the proposed system model and briefly outline the contributions.

PDSP-BENCH *for systematic...*

1. We emphasize the necessity for systematic performance modeling to understand how various factors—such as parallel data flow patterns, DSP system architecture, workload characteristics, and varying cloud resource configurations—influence PDSP performance. In this context, we delve into methods and frameworks essential for comprehensively capturing the complexities inherent in PDSP to address initial challenges. It provides detailed guidance for users to effectively define and execute parallel data flows across diverse hardware resources. Additionally, it facilitates the generation and corpus of valuable data to be used for training performance prediction models using machine learning using metrics like end-to-end latency. We introduce PDSP-BENCH, a novel benchmarking system designed specifically for performance modeling of parallel and distributed stream processing across heterogeneous hardware environments. It stands out by offering the most extensive

collection of benchmarks tailored for parallel dataflows, including *14 real-world* applications and *9 synthetic* workloads. This suite not only covers standard stream processing but also incorporates user-defined operators to ensure a comprehensive representation of DSP scenarios, rigorously designed to assess the scalability of DSP systems on heterogeneous hardware. In addition, PDSP-BENCH features a user-friendly web interface that supports automatic exploration of both real-world and synthetic parallel query structures across varied workloads and hardware configurations, providing a nuanced view of system heterogeneity, simplifies the testing as well as machine learning deployments (cf. Chapter 4).

...understanding of performance and benchmarking.

2. In the dynamic and heterogeneous environment of DSP systems with varying workloads (queries and data streams) and resource configurations make it impractical to execute all possible configurations to predict performance and determine initial optimal parallelism for PDSP. Traditional methods typically rely on manual tuning or online adaptive methods to achieve optimal parallelism, aiming to meet desired QoS. However, manual tuning results in varying configurations, often resulting in costly operator migrations and state management. In addition, these methods can lead to multiple oscillations in parallelism or require extensive data collection and iterative retraining for each new application, which becomes impractical with unseen workloads or changing resource landscapes. To address these inefficiencies, we propose ZEROTUNE, a novel performance or cost model that determines initial parallelism degrees based on predicted performance with query execution for seen and unseen workload and resource characteristics, significantly reducing the need for costly adjustments during the early stages of query execution. ZEROTUNE is designed to forecast performance accurately and generalize effectively across diverse streaming workloads and resource configurations, enhancing the adaptability and operational efficiency of DSP systems. It leverages a novel learning paradigm incorporating advances in transfer learning, specifically, data-efficient zero-shot learning. This innovative approach enables ZEROTUNE to adapt to varying DSP system dynamics offline, making it applicable across a broad spectrum of queries and configurations without the necessity for extensive retraining (cf. Chapter 5).

ZEROTUNE *for...*

...performance prediction and...

...determining parallelism.

3. Finally, the critical aspect is validating the proposed performance model and parallelism tuning methods in a real-world parallel and distributed stream processing environment. For this, we perform a comprehensive evaluation to assess the efficacy and applicability of performance modeling approaches within DSP systems. This evaluation is performed to establish robust criteria and methods that measure our proposed methods' performance, adaptability, and generalizability within the complex

Extensive evaluation in...

...real-world environment.

landscape of PDSP. The main aim is to ensure that the proposed models meet theoretical expectations and perform effectively in practical scenarios across various streaming workloads and computational environments offered by cloud resources. For instance, we expand the PDSP-BENCH system to include specialized workload and resource configurations designed to reflect the dynamic and heterogeneous nature of modern DSP operations, making it an essential tool for comparative analysis and a deeper understanding of different modeling approaches. Similarly, a significant part of our evaluation focuses on ZEROTUNE, explicitly examining how well performance models generalize and adapt across varying DSP workloads (query and data streams) and resource configurations (cf. Chapter 6).

4

PDSP-Bench: Benchmarking of Distributed Stream Processing

Recent advancements in distributed stream processing (DSP) have introduced various DSP systems, such as Apache Flink [52] and Storm [236], for analyzing data streams in real-time. These systems are essential for many modern data-driven applications to handle immense data volumes. For example, Netflix employs *Apache Flink* to handle more than 1.3 TB of data every day, requiring numerous parallel operator instances to manage the high influx and processing of data tuples [33]. For this, DSP systems offer a *data flow* abstraction to specify operator parallelism in the query and provide data partitioning strategies to manage data stream partitions. While such *parallel data flows* have become an intrinsic part of every DSP system, there is no means of systematic understanding of the performance of DSP under massively parallel dataflows (cf. RC1 in Section 1.1). To solve the first research challenge on performance understanding, we present a novel solution PDSP-Bench for systematic understanding of performance for the parallel and distributed stream processing (PDSP) in DSP systems.

DSP tackles the need of...

...data-driven applications.

Figure 13 illustrates the system architecture of PDSP-Bench for performance benchmarking and addressing the research challenges *RC1* (cf. Section 1.1) to understand the performance of PDSP. At the top level, continuous data streams are generated by various applications, such as click stream analytics and health monitoring. These data streams are then channeled through data stream connectors, which facilitate continuous data transfer to stream processing systems. Within these systems, the data is processed according to a defined operator graph, representing the queries executed on cloud resources. However, the complexity and variability inherent in such heterogeneous environments make it challenging to understand and evaluate the performance systematically [56, 79, 151]. PDSP-Bench aims to address this issue, offering performance modeling and benchmark systems to analyze and understand the performance of DSP systems for parallel and distributed stream processing.

Diversity and heterogeneity of environment...

...is a big challenge to understand performance.

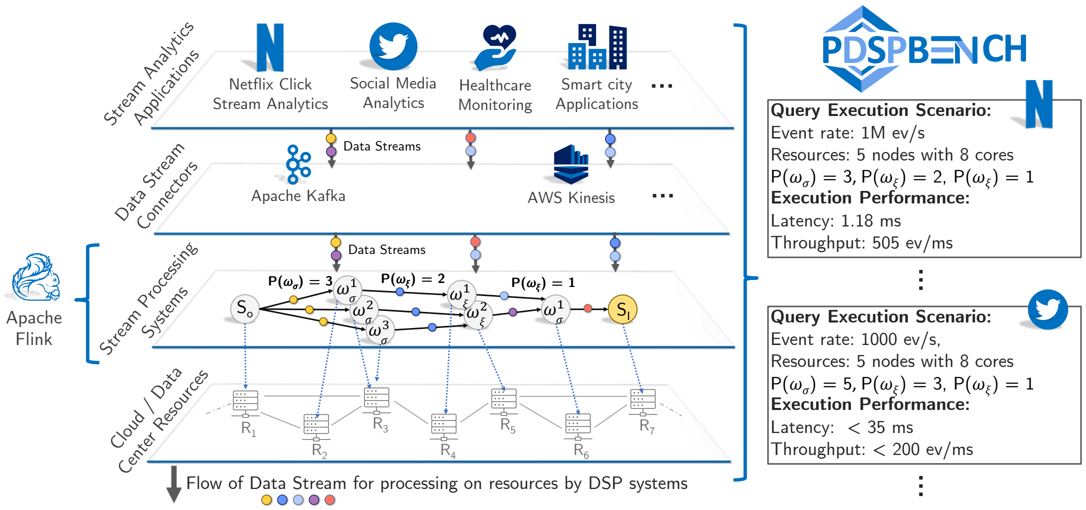


Figure 13: Overview of the PDSP-Bench architecture for performance benchmarking of parallel and distributed stream processing (PDSP) in DSP systems using various DSP workloads and resource configurations from various applications.

4.1 Analysis of Performance Benchmarking Systems

Existing benchmarking systems lack...

Most of the existing benchmarking systems for DSP are tailored towards the understanding of sequential dataflows [20, 64, 250, 132, 79, 197, 56]. Those benchmarking parallel dataflows [238, 91, 42, 267] are restricted to a homogeneous environment for resource placement and offer limited capabilities in terms of scaling workloads, e.g., event rate and query parameters like window length. We believe that a thorough analysis of parallel data flow graph placement on heterogeneous resources will reveal interesting insights into the behavior of distinct operators on various hardware resources and vice versa. Another unique aspect of our work is the ability to scale workload generation – both data streams and queries – by offering a benchmarking platform, which in fact can also be used for machine learning of DSP workloads, becoming increasingly important nowadays [107, 265]. In summary, we identify three primary challenges for PDSP-Bench by analyzing key existing benchmarking systems for DSP workloads presented in Table 3.

...support for parallel dataflows...

C1: Lack of expressiveness. Most existing benchmarks [64, 20, 250, 121, 133] often overlook the importance of benchmarking parallel dataflow applications, thus focusing only on sequential dataflows with a limited set of operators. For instance, *StreamBench* [162] overlooks essential operators, such as window functions, crucial for concurrent partitioning and efficient resource utilization.

Benchmark System	C1: P/S	C2: He/Ho	D/C	Infrastructure	C3: Learned DPS Models	Application Suite		Scalability
						Real-world	Synthetic	
Linear Road [20]	S	Ho	C	Single machine	No	1	-	No
YSB [64]	S	Ho	C	Single machine	No	1	-	No
StreamBench [162]	S	Ho	D	VMs	No	-	7	Partially
RIoTBench [214]	S	Ho	D	VMs	No	4	-	No
OSPBench [238]	S	Ho	D	Cloud AWS EC2	No	-	1	No
HiBench [121]	S	Ho	D	Local Cluster	No	-	4	No
BigDataBench [250]	S	Ho	D	Local Cluster	No	-	1	Partially
ESPBench [111]	S	Ho	D	VMs	No	5	-	No
SPBench [91]	P	Ho	C	VMs	No	4	-	Partially
DSPBench [42]	P	Ho	D	Azure Cloud Cluster	No	13	2	Partially
PDSP-Bench	**S/P**	**He/Ho**	**C/D**	**CloudLab, Geni Cluster, On-premise**	**Yes**	**14**	**9**	**Fully**

Table 3: Comparison of the existing benchmarking system for DSP with PDSP-Bench emphasizing the research challenges. Our work can effectively benchmark both parallel data flow graphs and heterogeneous hardware as well as can be used as a benchmarking system for training ML models on DSP workloads. Abbreviations used are S: Sequential plans, P: Parallel plans, He: Heterogeneous hardware, Ho: Homogeneous hardware, D: Distributed cluster, and C: Centralized or single machine.

C2: Shift to heterogeneity. The shift towards distributed and heterogeneous hardware requirements for benchmarking requires complex resource management, i.e., the underlying system must manage parallel resource mapping on varied hardware architectures, network links, and storage. Although benchmarking systems exist that assess parallel dataflows, like *DSPBench* [42] and *SPBench* [91]. These benchmarks are restricted to homogeneous hardware, reducing their relevance as real-world workloads often require heterogeneous environments [267]. For instance, Netflix runs on $1400+$ nodes on $50+$ distinct clusters with varied CPU cores [33] to deal with their demands of massively parallel dataflow applications.

...focus on homogeneous environment...

C3: Integrating learned DSP models. Most importantly, the rapid advancement in DSP mechanisms using machine learning (ML) necessitates a scalable and resource-friendly benchmarking system, ensuring its long-term relevance and utility in assessing future DSP with learned components. Recently, ML has been successfully applied for cost-based optimizations in DSP to support heterogeneous placements [107, 106] and deciding parallelism strategies [265, 8] and showed promising performance. This increasing surge of development in learned DSP models calls for a benchmarking platform that allows fair comparison among them by integrating the models and generating consistent training data for them. However, existing work do not provide

...and no integration of ML.

a means to integrate ML models such that they can be compared in a “fair” way with consistent metrics (cf. Table 3, Learned DSP models).

PDSP-Bench *tackles these challenges*

...by supporting various DSP workloads...

...heterogeneous configurations...

...cost models for benchmarking.

To solve these challenges, we propose PDSP-Bench [7], a novel benchmarking system specifically designed to tackle the three primary challenges faced by existing benchmarks: lack of expressiveness in benchmarking parallel dataflows, the necessity for heterogeneous hardware support, and the integration of learned DSP models. Unlike existing benchmarks, PDSP-Bench enables the creation and evaluation of parallel query structures (PQP) across a diverse range of operators and input data streams, which we divide into synthetic and real-world workloads, thus offering an expressive and scalable solution. We also provide mechanisms to configure and manage heterogeneous hardware resources by integrating resources from testbeds like CloudLab [76] with different configurations, which are essential for accurately reflecting real-world deployment scenarios. Furthermore, PDSP-Bench facilitates the integration of learned DSP models, allowing for systematic training and evaluation of these models on diverse streaming workloads. This integration is increasingly important given the surge of use of ML for optimizing DSP performance [8, 106, 265]. The system’s ability to generate large corpora of streaming datasets ensures that the ML models are trained on data representative of actual streaming workloads (cf. Section 4.3). The results show the importance of considering both parallelism and heterogeneity to achieve optimal performance in real-time data processing applications (cf. Section 6.2).

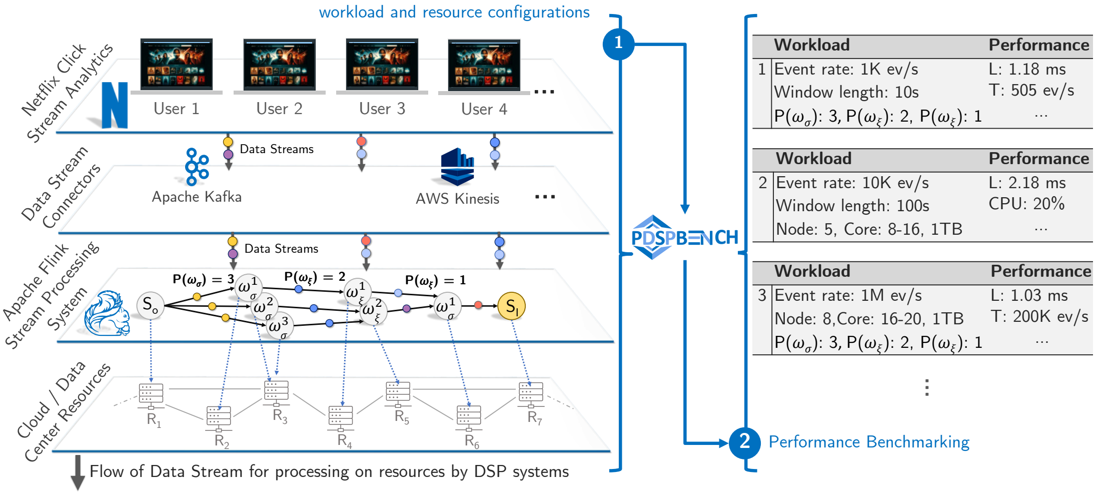
4.2 Problem Formulation for Performance Benchmarking

In this section, we present an extension of the overall performance modeling system specifically in the context of performance benchmarking described in Chapter 3: Section 3.2.1. We discuss the performance benchmarking scenario in Section 4.2.1 followed by benchmarking problem statement in Section 4.2.2.

4.2.1 *Performance Benchmarking Scenario*

Benchmarking Netflix click streams

In this section, we describe the need for a performance benchmarking system based on the possible real-world scenario of *click stream analytics* [82]. Netflix uses a *personalized recommendation* system [33] for enhancing user engagement by suggesting content based on personal viewing habits and preferences as presented in Figure 14. This system depends heavily on the real-time processing of vast amounts of data from millions of users, encompassing viewing history, search queries, and interactions. To manage this efficiently, Net-



	Workload	Performance
1	Event rate: 1K ev/s Window length: 10s $P(\omega_\sigma)$: 3, $P(\omega_\xi)$: 2, $P(\omega_\xi)$: 1	L: 1.18 ms T: 505 ev/s ···

	Workload	Performance
2	Event rate: 10K ev/s Window length: 100s Node: 5, Core: 8-16, 1TB	L: 2.18 ms CPU: 20% ···

	Workload	Performance
3	Event rate: 1M ev/s Node: 8,Core: 16-20, 1TB $P(\omega_\sigma)$: 3, $P(\omega_\xi)$: 2, $P(\omega_\xi)$: 1	L: 1.03 ms T: 200K ev/s ···

⋮

Figure 14: Netflix click stream analytics scenario demonstrates the use of parallel and distributed stream processing (PDSP) in DSP systems, such as Apache Flink. These DSP systems process various workloads (data streams and queries), on cloud cluster nodes with varying hardware configurations, such as those provided by CloudLab. PDSP-Bench benchmarks the performance of DSP systems for these workloads and hardware setups. It helps identify optimal configurations to fulfill QoS requirements such as minimum latency and maximum throughput with efficient utilization of resources. An example of click stream analysis is provided using a query specified in the Listing 1.

flix necessitates a highly scalable and efficient DSP system such as Apache Flink [52], Apache Storm [236], and Heron [144]. The challenge lies in processing such extensive data streams promptly to update recommendations in real-time, underscoring the importance of a robust DSP system that can handle peak loads without compromising the quality of service (QoS) [254] requirements such as minimum latency and maximum throughput.

Finding DSP that fits to usecase

```
SELECT CustomerID, ContentID, COUNT(*) AS ViewCount, AVG(Rating) AS AvgRating
FROM UserInteractions
WHERE EventType IN ('Play', 'Pause', 'Rate')
AND TIMESTAMPDIFF(MINUTE, InteractionTime, NOW()) <= 60
GROUP BY CustomerID, ContentID;
```

Listing 1: Example query of click stream analytics, such as Netflix's real-time personalization and recommendations to track user interactions.

The performance benchmarking setup for evaluating Netflix's DSP system involves simulating varying workloads (data streams and queries) that reflect actual user activity as shown in Listing 1[4]. These streams include viewing data, user interactions, and profile updates, which are crucial for generating accurate content recommendations. The system uses complex queries,

Different workloads from different users

[4] For simplicity, we use SQL as a syntax for the query

Finding resources for processing is hard

particularly the *recommendation update query*, to process these streams and update user recommendations based on the latest interactions. The resource configuration for this benchmarking spans across global data centers, employing low to high-processing instances of resources to handle the intensive computational demands of machine learning algorithms that generate these recommendations. Performance metrics, such as end-to-end latency, throughput, and resource utilization, are central to the benchmarking process. Latency measures the time taken from data ingestion to recommendation update, throughput assesses the volume of data processed per unit time, and accuracy evaluates the relevance of recommendations provided to users.

PDSP-BENCH *can help Netflix...*

...in benchmarking various scenarios...

PDSP-BENCH facilitates this process by setting up the necessary environment to mimic real-world operations, including the deployment of the DSP system across varied cloud resource configurations to test different computational scenarios. A significant focus of the performance benchmarking is on *how the DSP system performs under various configurations and loads*. PDSP-BENCH allows Netflix to experiment with different setups, such as varying parallelism of operators to distribute and process data across varying computational cloud resources in data centers. By doing this, it tries to find the optimal configuration that minimizes latency, maximizes throughput, and improves resource utilization. For instance, testing how the system performs with new machine learning models or during unexpected surges in viewership can help Netflix ensure that the recommendation system remains robust and responsive even under challenging conditions.

...and finding optimal configurations.

This detailed benchmarking process provides critical insights that help Netflix systematically understand DSP system performance for parallel and distributed stream processing and make informed decisions about resource allocation, system scaling, and potential optimizations. These decisions are essential for maintaining a high-performing recommendation system that can adapt to changing user behaviors and preferences, ensuring a personalized viewing experience that keeps users engaged and satisfied with the service. By rigorously testing different aspects of the DSP system, PDSP-BENCH plays a pivotal role in helping Netflix continuously improve its service delivery and operational efficiency.

4.2.2 *Benchmarking Problem Statement*

Consider Netflix's personalized recommendation scenario where each operator $\omega \in \Omega$ represents a part of the recommendation query, such as data preprocessing, user preference prediction, or content filtering. The goal is to optimize the recommendation system so that it can provide timely suggestions during peak hours without overloading the server resources (CPU, memory, and network). The decision variables will be set by solving the optimization problem to ensure that we have an optimal number of active operators P_ω at every time step t, such that resource capacities are not exceeded, and the performance requirements are satisfied, allowing the system to dynamically scale in response to workload changes, maintaining efficiency and quality of service. The multi-objective nature of this problem requires balancing the various goals, which may sometimes be conflicting. Therefore, it may be necessary to apply multi-objective optimization techniques or prioritize objectives according to business needs (e.g., prioritize latency over throughput during new releases).

How to benchmark...

...diverse workload...

Performance benchmarking in the context of DSP systems is the systematic evaluation of performance and analyzing DSP system's capabilities to process continuous and unbounded data streams D across a variety of parameters, including data streams (D), operator configurations (Ω), query parameters (Q), and resources (R). The benchmarking process is aimed at identifying the optimal configuration to optimize key performance indicators such as end to end latency (L), throughput (T), and resource utilization (U) under varying workloads and resource configurations. Each operator ($\omega \in \Omega$) has an associated parallelism degree $p \in P$, which defines the number of instances the operator is replicated across the system's resources $r \in R$.

...resource configurations...

...to find optimal configuration...

Performance benchmarking aims to understand different configuration and corresponding cost factors such as latency, throughput and resource utilization (cf. Definitions 6 to 12) . This understanding of the performance can be beneficial to identify the configuration that optimizes overall cost for specific scenario. For instance, benchmarking analysis can help to find an optimal configuration that minimizes overall cost (C)- the maximum possible throughput T and minimal possible end-to-end latency T while maintaining efficient resource utilization U across CPU, memory, and network. These performance metrics are crucial for maintaining the QoS and ensuring a responsive system as shown below:

...and to minimize the overall cost.

$$C = \operatorname*{arg\,min}_{C_L, C_T, C_{\text{CPU}}, C_{\text{mem}}, C_{\text{nw}}} \left(f(C_L, C_T, C_{\text{CPU}}, C_{\text{mem}}, C_{\text{nw}})\right) \tag{6}$$

$$f(C_L, C_T, C_{\text{CPU}}, C_{\text{mem}}, C_{\text{nw}}) = w_1 \cdot C_L + w_2 \cdot \frac{1}{C_T} + w_3 \cdot C_{\text{CPU}} + w_4 \cdot C_{\text{mem}} + w_5 \cdot C_{\text{nw}} \tag{7}$$

where $w_1, \ldots, w_5$ are weights that represent the relative importance of each metric.

- *Resource Constraints:* It ensures that the resource utilization cost does not exceed the available capacity for CPU, memory, and network bandwidth.
$$\sum_{\omega \in \Omega} C_{\text{res}}(\omega, t) \leq \text{Res}_{\max} \quad \forall \text{res} \in \{\text{CPU}, \text{mem}, \text{nw}\}, \forall t \tag{8}$$

- *Parallelism Constraints:* The parallelism degree of each operator must align with the system's ability to manage and distribute workload effectively.
$$P_{\min} \leq P(\omega) \leq P_{\max} \quad \forall \omega \in \Omega \tag{9}$$

- *QoS Constraints:* The DSP system must satisfy predefined QoS requirements, such as bounds on latency and minimum throughput for each operator.
$$L(\omega, t) \leq L_{\max}, \quad T(\omega, t) \geq T_{\min} \quad \forall \omega \in \Omega, \forall t \tag{10}$$

- *Windowing Constraints:* Window sizes for operators must fit within the allowed ranges based on the nature of the queries.
$$W_{\min} \leq W(\omega) \leq W_{\max} \quad \forall \omega \in \Omega \text{ with windowing} \tag{11}$$

4.3 PDSP-BENCH Design

PDSP-BENCH offers a performance benchmarking system to close the research gap *RC1* (cf. Section 1.2) related to the systematic understanding of DSP in a heterogeneous distributed environment. In the following section, we provide overview of PDSP-BENCH system components.

4.3.1 *Overview*

Large corpora of PDSP data using PDSP-BENCH

The main goal of PDSP-BENCH is to enable benchmarking of DSP systems considering heterogeneous environments for query deployment. As such, PDSP-BENCH aims to enable the creation of large corpora of streaming datasets across three dimensions: *query*, *data* and *resource* diversity. Such large corpora of datasets can be used in training ML models for learning optimizations of DSP such as cost of executing streaming queries and their placement on heterogeneous hardware. We demonstrate this by training and evaluating learned cost models using the dataset generated by PDSP-BENCH.

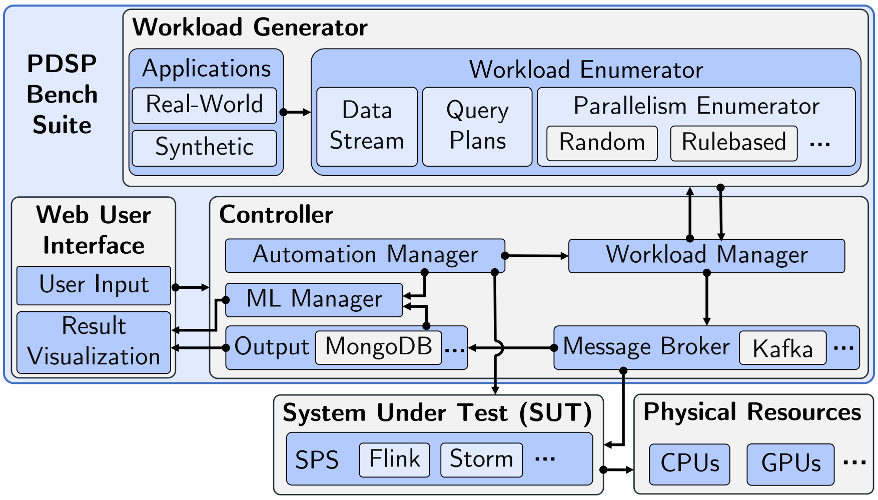


Figure 15: PDSP-Bench system overview.

While PDSP-Bench supports both sequential and parallel query plans (PQPs)[5], we mainly focus on PQP to show our novel contributions to tackle challenges. PDSP-Bench has three main components: (1) workload generator, (2) controller and (3) web user interface (cf. Figure 15). System under Test (SUT) represents the underlying Stream Processing System (SPS) like Apache Flink or Storm that are being evaluated by PDSP-Bench. We present an overview of our solution (**S#**) towards the goal (cf. RG1 Section 1.2) and show how we address the aforementioned challenges (**C#**) of existing work using PDSP-Bench components as follows.

C1: Lack of expressiveness. S1: To specify PQP with a wide range of operators and input data streams, PDSP-Bench provides a core component known as workload generator as shown in Figure 15. The task of this component is to enumerate various factors of the workload, including both data and query elements. For instance, it generates meaningful PQP by varying the *parallelism degrees* to be executed on the SUT, such as Flink. This approach enables both *query* and *data* diversity. These inputs on the enumeration can be given by the user via the web user interface that is managed by the controller as discussed later (cf. Section 4.5), but can also be configured directly into PDSP-Bench. A *key* issue we solve thereby is to generate PQP that are both *valid* and *representative* of current streaming applications. Thus, PQP must represent both standard streaming and user-defined operators that we selected from open-source data stream processing datasets like DEBS Grand Challenges [71]. We believe a combination of synthetic and real-world workloads is necessary to properly assess SUT's performance and generate

Support for PQP workloads

[5] By parallel query structures (PQP), we mean a given query structure with parallelism degrees that can generate multiple queries of this type of structure, e.g., linear PQP will generate a plan with parallel instances of filter operators with random filter literals.

datasets that are representative for ML in streaming platforms. We discuss this component in Section 4.4.

C2: Shift to heterogeneity. S2: We provide interfaces to the users to configure hardware resources that are used in turn to execute the generated PQP workload created by the former component. The controller and Web User Interface (WUI) components alleviate this complexity of configuring different hardware and hence enabling *resource* diversity for query execution and their deployment by automating it. We support the evaluation of heterogeneous CPU architectures, such as Intel and AMD, as well as distinct network, memory, and storage parameters by integrating the CloudLab cluster [76]. Additionally, other cloud providers can be easily integrated. Thus, the complex mechanism of setting up machines and query deployment using hefty resource providers like Kubernetes and Yarn in SUT is hidden using these components.

Diverse and heterogeneous resources

C3: Integrating learned DSP models. S3: The entire benchmarking system design holistically guides the users to specify PQP and its properties as well as their execution on different hardware resources that, in turn, can be used to generate data to train and evaluate ML models. For instance, PQP execution data can be used as features together with the performance metrics as labels, such as end-to-end latency, on a given SUT to train a *cost model* that predicts those metrics. Moreover, controller component allows integration of different ML models to support training on different sizes of Stream Processing (SP) workloads. To evaluate models, we report metrics such as accuracy (q-error) and training overhead (queries and time) as well as investigate trade-offs between them.

Cost models benchmarking

As a solution, we present the PDSP-BENCH workflow that shows how to use PDSP-BENCH (cf. Figure 15) to generate streaming workloads that can be used to train ML models and in turn also be used to predict the performance of a PQP from PDSP-BENCH using the trained model. All the user inputs are collected using the WUI (cf. Section 4.6) that are forwarded to the controller to orchestrate the benchmarking process. It allows users to select from existing applications in the suite (real-world or synthetic), but also provides a means to create novel applications in the form of PQP. Moreover, we provide other input parameters like parallelism enumeration strategies, workload and execution parameters such as event rate and query execution time (to limit the query as they are long-running) explained in the next Section 4.5. We also allow to store the generated workload in a database, e.g., MongoDB, that can be used for training ML models. Thus, ML Manager in the controller uses the "same" training data to train available ML models, e.g., a cost model can be trained to predict the costs of a PQP. This integrated approach allows "fair" comparison between ML models using our reported metrics such as training overhead. Thus, the reporting of benchmarks must also support training- and inference-related metrics and not just performance metrics. During the exe-

Support of synthetic and real-world applications cution of PQP on the selected SUT, the performance as well as the training metrics can be visualized in real-time on WUI.

We will focus on the workload generator component that is core to the PDSP-BENCH system in the following Section 4.5.

4.3.2 *Benchmarking Workflow*

The PDSP-BENCH offers simplistic steps for benchmarking DSP under SUT using `web user interface (WUI)`, guiding users through different options to provision and configure different hardware resources (cf. Table 7) following by specifying DSP for SUT to be benchmarked. This initial setup phase is crucial, as it determines the hardware configuration where the DSP will operate. These configurations are conveyed to the PDSP-BENCH's `controller` for orchestrating the benchmarking process, leveraging the `Automation Manager` to provision and prepare the selected hardware resources for SUT. This preparation includes the deployment of the DSP on the specified hardware, ensuring the execution environment for the benchmarking tasks. Following the environment setup, *WUI* offers users to choose from various applications, corresponding parallel query plans, and parallelism enumeration strategies, including configuring execution and workload parameters such as event rates, query execution time, and iterations. The `Automation Manager` orchestrates the `Workload Manager` to fetch and execute the selected parallel query plans on the SUT following triggering `message brokers` such as *Apache Kafka* [228] to create `input` and `output` topics for streamlining the flow of data streams and output data for chosen applications.

Ease to use

Controller is the orchestrator

WUI to take user input and show results

Post-execution, PQP output is forwarded to the `output` topic for validation and representation of results on *WUI*. Moreover, PQP and workload configuration, corresponding performances, are sent for real-time performance visualization and also stored in a database for comparative analysis of the performance of multiple configurations and clusters. Furthermore, PDSP-BENCH harnesses the machine learning mechanism to augment the benchmarking process further. The PQP and corresponding data in the database can be accessed by the *ML Manager* to generate training data representations and labels followed by training and inference using available ML models, such as DNNs and Random Forests (cf. Section 2.2.2). This holistic approach not only simplifies the benchmarking process but can also be leveraged by machine learning to optimize performance and improve benchmarking outcomes under various scenarios.

Data generation and ML integration

4.4 Workload Generator

An important research question we answer in this work is *"how to systematically generate workloads (data and query) for a comprehensive suite of dataflows to benchmark parallel and distributed streaming capabilities of SUT"*. The workload generator component plays a pivotal role in this question, as it generates data streams and PQP derived using an enumerator (Section 4.4.1) for our integrated synthetic and real-world applications aiding to benchmark any given SUT (Section 4.4.2)[6]. While we generate workload by varying parameters related to data, query and resources given in Table 7, e.g., event rates of upto 4 million events per second and parallelism degrees upto 128, they are in practice limited by the amount of resources which are available (e.g., the CloudLab cluster nodes m510, c630 and c6525_25g). Thus, the scale of workloads that PDSP-Bench can be much higher given the availability of the high amount of resources. In the following, we focus on how we diversify across these parameters.

Generate suite of parallel data flows

4.4.1 *Workload Enumerator*

This component enables *data* and *query* diversity by orchestrating the generation and distribution of a variety of data streams and PQP as described in the following.

Data stream: For synthetic applications, a common strategy to generate synthetic data is to *randomly* select from a given valid data range to avoid exhaustive enumeration of the given parameters, which is extremely time-consuming and practically impossible to do within a reasonable timeframe. In fact, *domain randomization* [230] is a common technique used for synthetic data generation to train ML models like deep neural networks such that it learns from the features of interest. The rationale behind this method is to have variability in the data so significant that the models trained on this data could generalize to the real-world data with no additional training [231]. Thus, to address this, PDSP-Bench includes a method for generating synthetic data streams by randomly varying over (1) tuple width (# data items in a single tuple of a data stream), (2) its data types (the data type per data item), and (3) event rates (# event tuples produced per time unit) crucial both for rigorously testing SUT's capabilities and collecting meaningful training data for ML models (cf. Table 7 for defined data ranges). While we generate data using the defined range, these values are highly configurable in the PDSP-Bench. To enable data streams from real-world applications, we use `Kafka`

Real-world scenario for producing...

...diverse data stream from various applications.

[6] Selected data: `https://github.com/pratyushagnihotri/pdsp-bench_experiment_data`

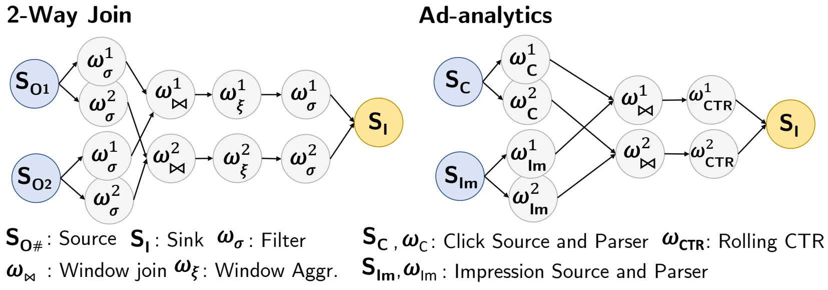


Figure 16: Example of PQP for synthetic: 2-way join and real-world: ad analytics application.

as a data producer that is connected via PDSP-Bench to the SUT. We *repeat* the data stream read from the source to mimic infinite data streams.

Query plans: For synthetic query plans, we offer an extensive range of PQP from an array of query structures, including simple linear queries with one filter to complex configurations involving multi-way joins and multiple chained filters. To give an example representation of such a 2-way join, see Figure 16 (left). Moreover, we randomly enumerate over multiple parameters of these query structures, such as filter function (e.g., $<$, $\leq$), its data type (the filter value's data type), window type (e.g., sliding, tumbling), window policy (e.g., time, count), etc., to generate PQP that can be again used to evaluate a given SUT and is representative of workloads required to train a ML model. While we generate these query parameters randomly, an important question arises: how do we balance query properties, e.g., selectivity? For instance, a random selection of filter literals may result in data never passing the generated filter. To avoid this, we use selectivity estimation methods [8] to estimate the selectivity of a given filter operator such that queries with only valid literals are generated, where $sel_{\omega_\sigma}$is not 0. For real-world query plans, we enable users to choose from the given range of applications available in our benchmark suite but also use them as a basis PQP to generate more queries. For instance, consider the ad analytics application in Figure 16 (right), where the users can choose to generate more query plans by adding a filter operator, choosing a different window count for the join, etc. This way, we allow users to execute given applications but also generate PQP to evaluate SUT capabilities for dynamics and uncertainty inherent to real-world applications. Moreover, this flexibility allows the generation of representative PQP aligned to the real world to train ML models. The full list of query applications is described in Table 4 and explained in Section 4.4.2. Other data ranges to configure query plans available in PDSP-Bench can be seen in Table 7.

Extensive range of simple and complex PQP

Benchmarking about standard operators and UDO

Parallelism enumerator: While random enumeration is meaningful as it represents real-world data ranges for the parameters discussed so far, e.g., tuple widths for data streams and window length for operators in PQP, we

Meaningful enumeration of parallelism...

...using different strategies.

note that random enumeration of parallelism degrees for operators is not the same. The random selection of parallelism degrees in PQP will result in very noisy queries or even invalid plans, e.g., selecting higher parallelism degrees for downstream operators is less meaningful since there are anyways less tuples that have to be processed as tuples move down in the data flow graph (e.g., after filter operator). Moreover, a random selection of parallelism degrees, e.g., $\omega_\sigma = 1$, $\omega_\bowtie = 10$ in the 2-way join example query in Figure 16 leads to a plan that is very bad in performance because it first limits processing capabilities by selecting only one instance of filter and hence there is limited use of 10 instances of join operators and highly wasteful of resources. While such bad plans might still be interesting for benchmarking SUT to cover corner cases, learning ML models with such bad plans is not meaningful as they are not encountered in real-world. Thus, we employ different strategies for parallelism degree enumeration in PDSP-Bench that can be selected by the user depending on the needs. For instance, we provide `Rule-based` strategy that selects *meaningful* parallelism degrees for PQP derived based on literature [131] but also random enumeration as defined below.

Benchmarking different enumeration strategies

`Random` selects a parallelism degree randomly within the given range, usually up to a maximum number of cores available on physical resources, introducing variability for comprehensive performance assessment of SUT. `Rule-based` goes beyond randomness and selects parallelism based on workload characteristics and physical resources. It considers factors such as event rates, operator selectivity, and the number of cores, enabling a more targeted enumeration of parallelism for upstream and downstream operators. This approach seeks to optimize performance by aligning parallelism with the specific demands and capacities of the system [131]. `Exhaustive` aims to test every unique combination of parallelism degrees, ensuring that each combination is tested. `MinAvgMax` cycles through generating queries with minimum, average, and maximum numbers of parallelism degrees, systematically exploring the effects of varying parallelism degrees on system performance, from least to most intensive use of resources. `Increasing` evaluates the impact of incremental change in parallelism, starting at the minimum degree and increasing stepwise to the maximum for each operator up the dataflow graph. `Parameter-based` is designed for rapid testing, as it configures parallelism based on user input.

4.4.2 *Applications*

We include a selection of applications in the PDSP-Bench benchmarking suite by analyzing previous research works in databases [64, 20] and stream processing [42, 111, 132, 91]. The applications are chosen based on a set of criteria that capture the diversity of streaming workloads, including data sources' tuple width, the data items' type, as well as the different operators and their complexity, e.g., standard DSP and user-defined operators in data flow graphs. To thoroughly assess PDSP in heterogeneous environments, we classified these applications into real-world and synthetic categories. This approach ensures a detailed evaluation of the SUT capabilities, with a special emphasis on its performance (latency and throughput) but also readiness for future demands across various conditions from typical to peak usage scenarios for ML integration.

Applications	**Area**	**Description**
Word Count (WC) [144]	Text Processing	Processes a text stream, tokenizes sentences into words, and counts the occurrences of each word in real-time using a key-based aggregation.
Machine Outlier (MO) [128]	Network Monitoring	Detects network anomalies in machine usage data streams using the *BFPRT algorithm* [262] to identify outliers based on statistical medians.
Linear Road (LR) [20]	Traffic Management	Processes vehicle-generated location data through four queries: *toll notification*, *accident notification*, *daily expenditure*, and *total travel time*, to calculate charges or detect incidents.
Logs Processing (LP) [218]	Web Analytics	Processes HTTP Web Server log data to extract insights using two queries: one counts visits within specified intervals, and the other tallies status codes.
Google Cloud Monitoring (GCM) [134]	Cloud Infrastructure	Analyzes cloud computing data by calculating average CPU usage over time, either grouped by job or category, with results processed through sliding windows and specific grouping operators.
TPC-H (TPCH) [39]	E-commerce	Processes a stream of order events to emit high-priority orders, utilizing operators to structure, filter, and calculate the occurrence sums of order priorities within specified time windows.
Bargain Index (BI) [32]	Finance	Analyzes stock quotes streams to identify bargains by calculating the price-to-volume ratio against a threshold using *VWAP* and *Bargain Index* Calculators, emitting qualifying quotes.
Sentiment Analysis (SA) [75]	Social Network	Determines the emotional tone of tweets by assessing sentiment using TwitterAnalyzer and SentimentClassifier operators, which apply Basic or LingPipe classifiers to score and label the tweets.
Smart Grid (SG) [70]	Sensor Network	Analyzes smart home energy usage through two queries that calculate global and local average loads using sliding window.
Click Analytics (CA) [162]	Web Analytics	Analyzes user interactions with online content through two queries: grouping click events by Client ID for repeat and total visits per URL, and identifying geographical origins using a Geo-IP database.
Spike Detection (SD) [215]	Sensor Network	Processes sensor data streams from a production plant to detect sudden temperature spikes by calculating average temperatures over sliding windows, and identify spikes exceeding 3% of the average.
Trending Topics (TT) [171]	Social Network	Processes stream of tweets using the TwitterParser and TopicExtractor operators to identify trending topics on Twitter based on aggregated popular topics based on predefined thresholds.
Traffic Monitoring (TM) [156]	Sensor Network	Processes streaming vehicle data using TrafficEventParser and RoadMatcher operators to match vehicle locations to road segments then calculates average speed per segment using the AverageSpeedCalculator.
Ad Analytics (AD) [181]	Advertising	Processes real-time data on user engagement with digital ads by parsing clicks and impressions, calculating their counts within time windows, and computing the click-through rate (CTR) with a rolling CTR operator.
Synthetic Queries	Standard DSP Queries	Assess standard streaming workloads by randomly generating diverse data streams and query structures with increasing complexity. It supports various data types and standard operators like filter, window aggregate, window join, and groupby to evaluate streaming operators through synthetic query structures, from simple linear to complex multi-join queries.

Table 4: Benchmarked parallel query structures based on synthetic and real-world applications (based on [123, 42, 111, 91, 267]).

Facilitates both synthetic and real-world application...

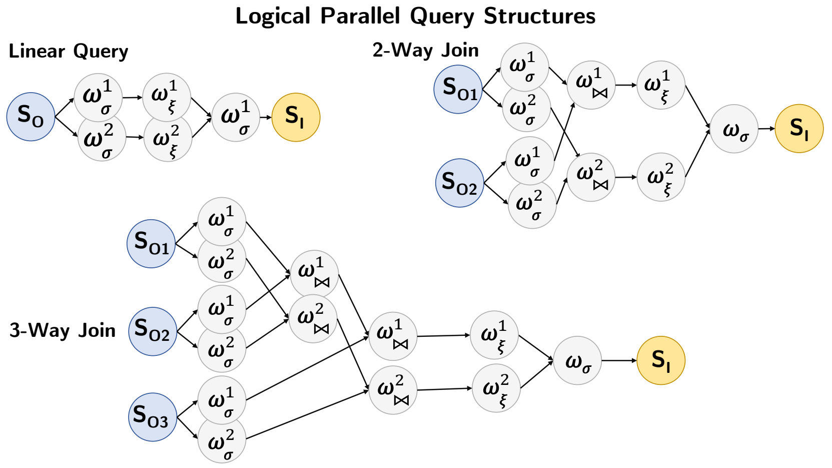


Figure 17: Examples of parallel query structures for synthetic applications are liner, 2-way joins, and 3-way joins. These structures are used to generate diverse parallel query plans (PQP) or parallel data flows. In addition, multiple chained filters and joins are used to generate more complex parallel query structures, such as 2-,3-, and 4-chained filters and 4-, 5-, and 6-way joins.

...for benchmarking and data generation.

The real-world applications reflect genuine data streams, such as *social media feeds*, *financial transactions*, and *IoT sensor data*, which are crucial for mimicking actual system loads and behaviors for the benchmarking process as presented in Table 4. For instance, DEBS 2014 Smart Grid data [214] serves as a real-world benchmark, reflecting energy usage patterns from smart plugs. On the other hand, synthetic applications also represent real-world scenarios by including standard DSP operators like filters, aggregates, and joins, but the data streams are generated artificially, allowing to stress test SUT under hypothetical future scenarios with high data volumes. This dual approach ensures a balanced assessment of SUT's performance, scalability, and adaptability, preparing it for current and future data processing challenges. We enlist all the (both synthetic and real-world) applications included in PDSP-Bench in Table 7 and briefly explain them in the rest of the section. While we provide the applications described above (cf. Table 4), PDSP-Bench can be easily extended by integrating new queries from other benchmarks like YSB [64] and Nexmark [237].

Synthetic Applications

Generate data stream...

The synthetic world application, with its primary purpose of evaluating different workloads, is specifically designed to handle varying data streams and query structures. These components play a crucial role in generating diverse parallel query plans or data flows, thereby increasing the complexity and variability of data distribution. In this application, the data tuples encapsulate various field values, labels, and a timestamp. These data tuples are classified into different data types, such as integers, doubles, and strings. Each data type allows for specific operations, reflecting their distinct data characteristics. For instance, strings can be operated with functions like startsWith, while numeric types can use operations like *less than or equal* (leq).

...different data tuples and types.

To simulate realistic scenarios, the application is structured into *nine* synthetic query models that range from simple to complex configurations such as *linear*, *two-way*, and *three-way* joins followed by facilitating these structures to generate diverse parallel query plans or parallel data flows as presented in Figure 17. Additionally, multiple chained filters and joins are used to grow the complexity of these parallel query structures, resulting in more complex data distribution patterns and configurations ranging from *2-filter*, *3-filter*, and *4-filter* setups to more intricate *4-join*, *5-join*, and *6-join*. Each model begins with a source and concludes with a sink, incorporating intermediary operators like *filters* (ω_σ), *window aggregates* (ω_ξ), and *window joins* ($\omega_\bowtie$). The presence and properties of these operators are randomly determined during query generation, adding to the variability of the system. For instance, the simplest query possible under the linear structure might only involve a *source*, a *window aggregate*, and a *sink*. Conversely, a more complex setup, such as depicted in a three-way join scenario, could include up to eleven operators, including various filters and joins.

Consisting filter, window join and aggregation operators

The Filter operator provides various functions for different data types. For strings, it includes *startsWith*, *endsWith*, *endsNotWith*, *startsNotWith*, and *contains*. Similarly, for integers and doubles, it supports $\neq$, $<$, $>$, $=$, $\leq$, and $\geq$. While the *window-Aggregate* operator only processes integers and doubles with functions like *sum*, *mean*, *max*, and *min*. A practical example involves executing a linear query sequence (Source -> Filter -> Window Aggregate -> Filter -> Sink) with fixed parameters, except for varying the parallelism degree of the involved filters and window aggregate operators. Here, the source operator consistently remains set to a parallelism degree of 10. The use of the *Key By* transformation, akin to a *Group By* in traditional databases, is crucial before a window aggregate to avoid limitations. Without *Key By*, data would funnel to a single task instance, severely restricting parallelism capabilities. For instance, a window aggregate calculation with *Key By* allows data distribution based on tuple values, enhancing load distribution and enabling grouped totals calculation. To further diversify query variations, the application considered replacing *Key By* with a splitter component to distribute data based on time or count. However, due to the deterministic nature required by systems like Apache Flink, this idea faced implementation challenges. Instead, a timestamp-based *Key By* approach was adopted, ensuring data can be deterministically split and processed across multiple windows and operators. This method also extends the range of key values by incorporating the

Various range of operator functions

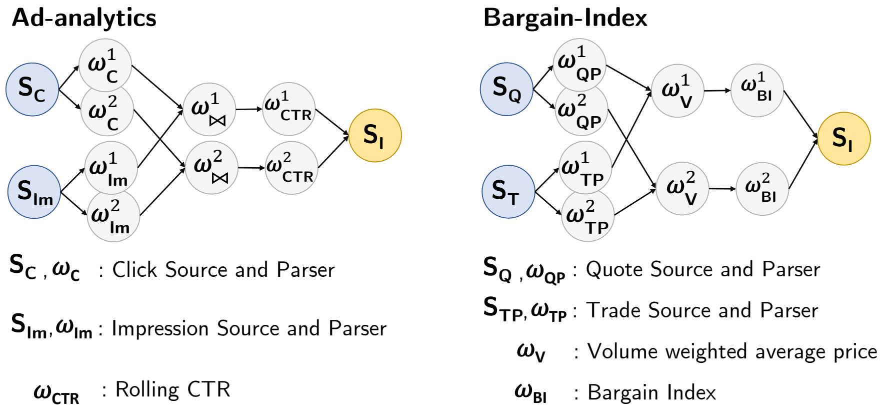


Figure 18: Example of parallel query structures for ad-analytics and bargain-index applications.

timestamp's last digit, reducing data skew and enhancing the system's ability to handle larger parallelism degrees effectively.

Real-world Applications

Applications from real-world scenario

PDSP-Bench supports 14 real-world applications from different industry domains. In the following sections, we describe the parallel query structure and operator type of these applications followed by their DAG to show the data distribution and data stream flow between these operators.

Multiple data streams for ad injection

Ad Analytics (AD): The Ads Analytics application [181] analyzes streaming data from user interactions with advertisements, aiming to derive actionable insights about advertisement performance in real-time. Initially, the data stream is split, directing it to two distinct processing streams. One stream passes through the `ClickParser` operator, which isolates and categorizes data points like clicks, views, and advertisement IDs, essentially transforming raw data into *click* events. Concurrently, the `ImpressionParser` operator handles the other stream, focusing on *impressions* from user interactions.

In Figure 18, each stream is processed by its respective counting operators— `ClickCounter` (ω_C) and `ImpressionCounter` (ω_{Im})—where data is accumulated over specified time windows. A join operator then merges these streams by synchronizing click and impression counts against their common query and advertisement identifiers. The aggregated data flows into the `RollingCTR` (ω_{CTR}) operator, which employs a sliding window mechanism to calculate the Click-Through Rate (CTR) by dividing the total clicks by total impressions within the window. The processed data, encapsulating CTR metrics, is then conveyed to the sink for further action or reporting.

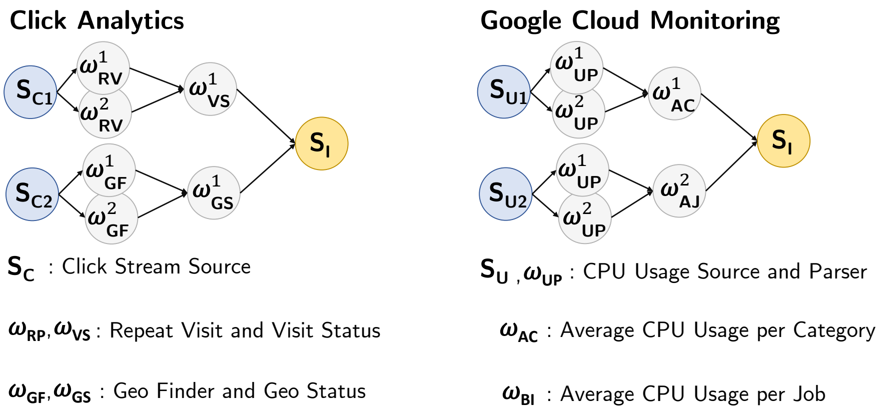

Click Analytics
S_C1
ω^1_RV
ω^2_RV
ω^1_VS
S_C2
ω^1_GF
ω^2_GF
ω^1_GS
S_I
S_C : Click Stream Source
ω_RP, ω_VS : Repeat Visit and Visit Status
ω_GF, ω_GS : Geo Finder and Geo Status
Google Cloud Monitoring
S_U1
ω^1_UP
ω^2_UP
ω^1_AC
S_U2
ω^1_UP
ω^2_UP
ω^2_AJ
S_I
S_U , ω_UP : CPU Usage Source and Parser
ω_AC : Average CPU Usage per Category
ω_BI : Average CPU Usage per Job

Figure 19: Example of parallel query structures of click analytics and Google cloud monitoring applications.

Bargain Index (BI): The Bargain Index application [96, 17, 68, 32] has been adapted from IBM InfoSphere Streams. It focuses on identifying stock quotes representing potential bargains by analyzing transactions from financial markets. It is crucial for the finance industry as it automates the detection of bargain opportunities in stock trading, enhancing decision-making processes by providing timely insights into undervalued stocks.

Stock trades analysis

In Figure 18, the application's workflow ingests streams of both trades and quotes, segregating them into two distinct paths for parallel analysis. In one path, the `Volume-Weighted Average Price (VWAP)` (ω_V) operator calculates the average price of trades for specific stock symbols based on the last 15 trades, adjusting for volume. Concurrently, the `BargainIndexer` (ω_{BI}) operator analyzes the quotes stream and computes a bargain index by comparing the price and volume of each quote against the most recently computed VWAP. Only quotes with a bargain index exceeding a predefined threshold are forwarded to the sink for further action.

Click Analytics (CA): The Click Analytics application [162] processes user interaction data from a website, primarily analyzing logs in the Common Log Format for real-time analysis of user behavior and geographic engagement patterns, facilitating a deeper understanding of user interactions and site engagement directly from web server logs. These logs need to be parsed to extract key data fields such as the timestamp, URL, and user IP address, which may serve as a user identifier in the absence of other identifiers. In Figure 19, `RepeatVisit` (ω_{RV}) operator groups events by URL and user ID, using these as keys in an associative array to check for repeat visits by the same user to the same URL. Concurrently, the `VisitStats` (ω_{VS}) operator counts both total and unique visits. Simultaneously, `Geography` operator (ω_{GF}, ω_{GS}) connects to a GeoIP database to ascertain the geographic location of users based on their IP addresses to extract city and country information from the location data

Users engagement analysis

and passes it on to the GeoStats operator. This operator maintains an associative array that logs visit counts per country and city, updating its records with each new event and emitting updated geographic statistics.

The application supports two queries. The first involves grouping click log events by client ID within a specified time window and identifying repeat visits by analyzing URL and client ID combinations. The results, including counts of total and unique visits for each URL, are forwarded to the sink. In the second query, the Geography operator determines the geographic origin of each click using a Geo-IP database. It matches IP addresses from the events to locations and generates GeoStats objects for each geographical unit, complete with detailed visit counts by city. This processed data stream then moves to the sink operator, completing the analytics pipeline.

Finding resource usage

Google Cloud Monitoring (GCM): The Google Cloud Monitoring [134] application processes and analyzes CPU usage patterns within a cloud computing environment to calculate average CPU usage over time, which can be grouped by job or category based on user-selected queries. In Figure 19, the `TaskEventParser` (ω_{UP}) operator extracts essential information from the raw data, structuring it into objects that include attributes such as timestamp, job ID, task ID, machine ID, event type, user ID, category, priority, and resource usage metrics like CPU, RAM, and disk usage.

For the first query, the data stream is directed to a sliding window operator that organizes events by category. Following this categorization, the `CPUUsagePerCategory` (ω_{AC}) operator takes over, calculating total CPU usage, the earliest timestamp in each window, and grouping these by category. This allows for an aggregated view of CPU consumption across different categories. In the second query, the stream from the `TaskEventParser` is routed to the `CPUPerJobCalculator` (ω_{AJ}) operator. This operator applies a sliding window to group events by job ID and computes the average CPU usage for each job. This calculation provides insights into CPU demand per job, aiding in resource allocation and performance monitoring.

Simulate roads tolling system

Linear Road Benchmark (LRB): The Linear Road Benchmark [20] is designed around a simulated tolling system on a motor vehicle expressway. In this benchmark, as presented in Figure 20, vehicles generate periodic location data as they travel along the road. This data stream forms the basis for four distinct queries: toll notification, accident notification, daily expenditure, and total travel time. Initially, `VehicleEventParser` (ω_{VE}) operator transforms the raw data into structured vehicle event objects. These objects include details such as vehicle ID, road segment, speed, and event type. The first query, the parsed data is sent to the `TollNotification Mapper` (ω_{TN}), which calculates tolls based on the number of vehicles and their average speed during the specified period. The results are then forwarded to the sink operator fol-

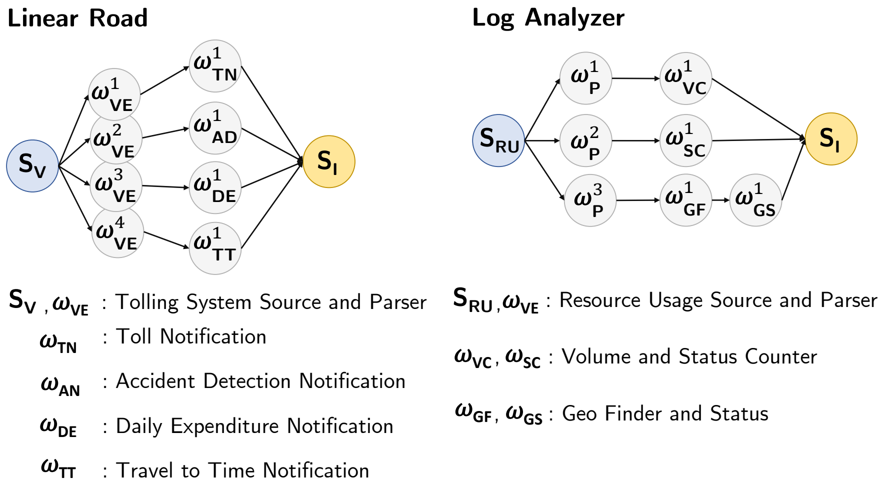

Linear Road
Log Analyzer
S_V
ω^1_{VE}
ω^2_{VE}
ω^3_{VE}
ω^4_{VE}
ω^1_{TN}
ω^1_{AD}
ω^1_{DE}
ω^1_{TT}
S_I
S_{RU}
ω^1_P
ω^2_P
ω^3_P
ω^1_{VC}
ω^1_{SC}
ω^1_{GF}
ω^1_{GS}
S_I
S_V , ω_{VE} : Tolling System Source and Parser
ω_{TN} : Toll Notification
ω_{AN} : Accident Detection Notification
ω_{DE} : Daily Expenditure Notification
ω_{TT} : Travel to Time Notification
S_{RU} , ω_{VE} : Resource Usage Source and Parser
ω_{VC}, ω_{SC} : Volume and Status Counter
ω_{GF}, ω_{GS} : Geo Finder and Status

Figure 20: Example of parallel query structures of linear road benchmarking and resource usage logs analyzer.

lowed by the second query, the `AccidentDetection` (ω_{AD}) operator receives data from the event parser and identifies accidents by detecting duplicate vehicle positions within a defined time window, signaling potential collisions or traffic disruptions. The third query involves calculating the daily expenditure for each vehicle using `DailyExpenditureCalculator` (ω_{DE}) operator to maintain state information about vehicle events and corresponding expenditures. Finally, the fourth query calculates the total travel time for each vehicle using the `TravelTimeNotification` (ω_{TT}) Mapper operator to analyze timestamps for determining the total duration of travel for each vehicle and sends this information to the sink.

Log Analyzer (LA): The Log Processing application [142, 218] processes streams of HTTP server logs formatted in Common Log Format to extract meaningful real-time insights, crucial for understanding traffic patterns and server request details on web servers. These logs include data fields such as the client's IP address, timestamp, request verb, resource name, and status code, as shown in Figure 20. Upon receipt, the logs are parsed to structure the necessary data, which is then directed simultaneously to three distinct operators. The `VolumeCounter` (ω_{VC}) operator tallies the number of visits, counting each log entry as a single visit within a specified minute window. The `StatusCounter` (ω_{SC}) operator maintains a tally of each HTTP status code encountered, storing these counts in an associative array. Meanwhile, the `GeoFinder` (ω_{GF}) operator with `GeoStatus` (ω_{GS}) operator utilizes the client's IP address against an IP location database, such as MaxMind or GeoIP, to determine and emit the geographical location—country and city—of the user.

Web server logs for anomalies detection

Additionally, the Logs Analyzer application, another stream processing setup, extracts data from similar HTTP web server logs. It features two main

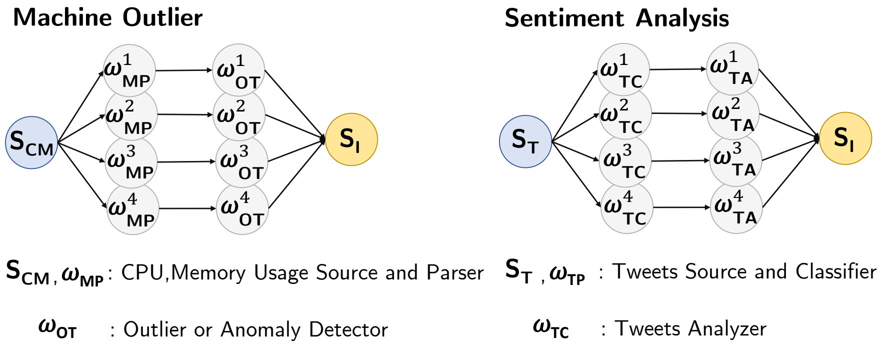


Figure 21: Example of parallel query structures of machine outlier and sentiment analysis.

operators: the `StatusCounter` and the `VolumeCounter`, which support two distinct queries. The first query uses the `VolumeCounter` to compute the number of visits within a given time frame, making it a stateful operator that increments with each new log event. The second query leverages the `StatusCounter` to aggregate counts of each status code during the same window.

Outlier detection from machine logs

Machine Outliers (MO): The Machine Outlier application [262, 128] processes network computer usage data to identify anomalies particularly effective in detecting significant deviations in machine behavior from typical patterns using the Blum-Floyd-Pratt-Rivest-Tarjan (BFPRT) algorithm, often referred to as the *median of medians* algorithm as shown in Figure 21. The `MachineUsageParser` (ω_{MP}) operator standardizes the raw input strings into a structured format and assigns default values to any fields that may be missing. Following parsing, the data stream advances to the Window Operator, which organizes the usage data into batches according to a predefined time window. This temporal grouping is crucial for analyzing data within specific intervals and ensuring the relevance of the anomaly detection process. Post-windowing, the data is transferred to the `BFPRTAlgorithm` (ω_{OT}) operator to determine the statistical median—or the kth element—within each windowed batch. Anomalies are identified by measuring the deviation of data points from this median value.

Users mood analysis from tweets

Sentiment Analysis (SA): The Sentiment Analysis application [75, 162, 42] processes a continuous stream of tweets formatted in JSON containing fields like tweet ID, timestamp, and text. It leverages a Natural Language Processing (NLP) technique to analyze the sentiment of textual content as illustrated in Figure 21. Specifically, it counts the number of positive and negative words within a sentence and uses their net difference to determine the overall sentiment polarity. Tweets are filtered out which are not in the default and currently supported language. Extending support to other languages can be achieved by loading the appropriate lists of positive and negative words.

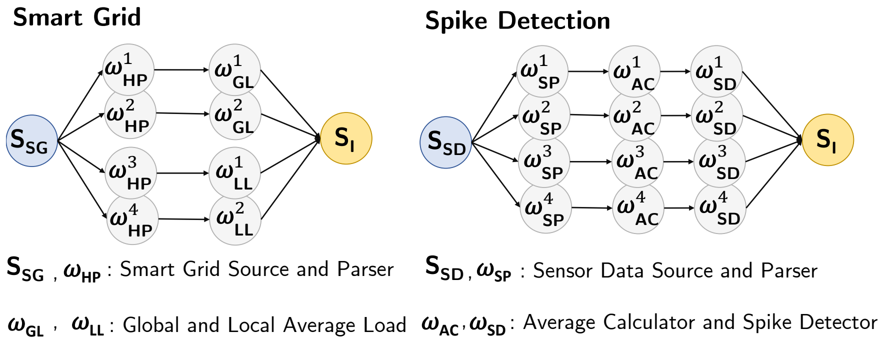


Figure 22: Example of parallel query structures of smart grid and spike detection.

Initially `Classifier` (ω_{TC}) operator removes *stop words*—words from tweets that typically carry no significant sentiment value followed by counting the positive and negative words within each tweet using `TwitterAnalyzer` (ω_{TA}) operator. Within the Twitter Analyzer, a Sentiment Classifier operator is employed, which can utilize either a Basic Classifier or a LingPipe Classifier. The Basic Classifier assigns sentiment scores based on a predefined sentiment map, whereas the LingPipe Classifier uses a pre-trained model for more nuanced sentiment scoring. Both classifiers label the tweets as negative, positive, or neutral and assign corresponding scores. Finally, the Twitter Analyzer encapsulates these scored tweet objects and forwards them downstream to the Sink, completing the sentiment analysis workflow.

Load detection using grid data

Smart Grid (SG): The Smart Grid application [214, 70] is designed to monitor energy consumption within a smart electricity grid as presented in Figure 22. Its main objectives include load prediction and outlier detection, generating two key outcomes: identifying outliers per house and predicting house/plug loads. Outliers are detected by comparing individual house plug medians with the global median, with values surpassing the latter considered outliers. Load prediction utilizes current average and median data to forecast future loads. In this context, the application employs two queries ① calculates the global average load of houses using `HouseLoadParser` (ω_{HP}) operator to first parse the input data stream into `HouseEvent` objects and grouping them using `house ID` and is subjected to a sliding window deriving the average load per house `GlobalAverageLoad` (ω_{GL}). Similarly, ② calculates the local average load `LocalAverageLoad` (ω_{LL}), factoring in attributes like house identifier, household count, and plug number.

Real-time monitoring from IoT sensor data

Spike Detection (SD): The Spike Detection application [215, 18] represents the real-time monitoring streaming sensor data from production plant machinery, specifically focusing on temperature measurements, crucial for identifying anomalies such as abrupt temperature increases, or spikes, that could indicate potential issues with machine operations as presented in Figure 22. Initially, the `SensorParser` (ω_{SP}) operator processes the incoming raw data

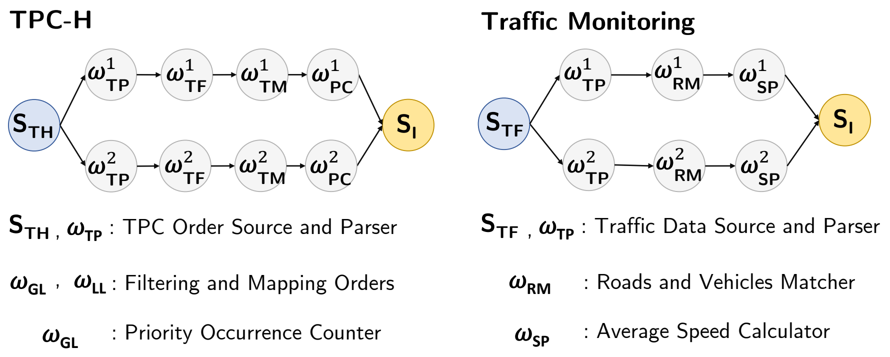

TPC-H
Traffic Monitoring
ω^1_{TP} ω^1_{TF} ω^1_{TM} ω^1_{PC}
S_{TH}
S_I
ω^2_{TP} ω^2_{TF} ω^2_{TM} ω^2_{PC}
ω^1_{TP} ω^1_{RM} ω^1_{SP}
S_{TF}
S_I
ω^2_{TP} ω^2_{RM} ω^2_{SP}
S_{TH} , ω_{TP} : TPC Order Source and Parser
ω_{GL} , ω_{LL} : Filtering and Mapping Orders
ω_{GL} : Priority Occurrence Counter
S_{TF} , ω_{TP} : Traffic Data Source and Parser
ω_{RM} : Roads and Vehicles Matcher
ω_{SP} : Average Speed Calculator

Figure 23: Example of parallel query structures of TPC-H and traffic monitoring.

stream to extract vital information such as sensor ID, timestamp, and temperature readings. This data is then structured into tuples, forming a well-organized stream that flows into the `AverageCalculator` (ω_{AC}) operator based on key-based sliding window aggregation to calculate the moving average temperature within a predefined window period for each sensor ID. Following this, the processed data advances to the `SpikeDetector` (ω_{SD}) operator that compares the current temperature against the calculated average to identify temperature spike when the current reading exceeds more than 3% of the average temperature. Only these spikes are passed downstream, filtering out all non-spike events.

Transaction analysis

TPC-H (TPCH): The TPC Benchmark-H [39] is a decision support benchmark that includes queries and database data with industry-wide relevance. It features business-oriented ad-hoc queries and concurrent data modifications, as shown in Figure 23. We selected one query from TPC-H benchmark that processes data stream to emit high-priority orders. The ① `TPCHParser` (ω_{TP}) operator parses the data into a structured format, such key, customer name, address, and order priority followed by ② `TPCHFilterCalculator` (ω_{TF}) filter operator to filter out orders with defined priority. This filtered data stream is directed to the ③ `PriorityMapper` (ω_{TM}) operator, which creates tuples of priority and occurrence then forwarded to the ④ `PriorityCalculator` (ω_{PC}) aggregation operator to compute the sum of occurrences for each order priority within a specified sliding time window.

GPS data for traffic congestion

Traffic Monitoring (TM): It processes events emitted by vehicles, such as vehicle identifier, GPS coordinates, direction, speed, and timestamp [156, 97, 23] as presented in Figure 23. The ① `TrafficEventParser` (ω_{TP}) operator structures the data into events. These events then pass to ② `RoadMatcher` (ω_{RM}) operator which determines the road each vehicle is on by initializing with a city's bounding box, filtering events outside city limits, and using a shapefile of city road followed by adding the road ID to the event and sends it to the ③ `AverageSpeedCalculator` (ω_{SP}) operator for computing the average speed over sliding time windows for each road along with the number of vehicles.

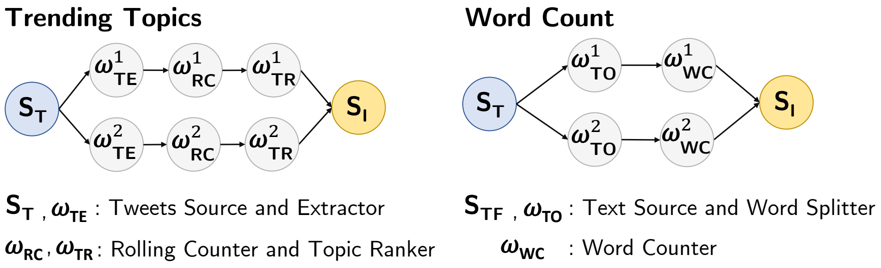

Trending Topics
S_T
ω^1_{TE}
ω^1_{RC}
ω^1_{TR}
ω^2_{TE}
ω^2_{RC}
ω^2_{TR}
S_I
S_T, ω_{TE} : Tweets Source and Extractor
ω_{RC}, ω_{TR} : Rolling Counter and Topic Ranker
Word Count
S_T
ω^1_{TO}
ω^1_{WC}
ω^2_{TO}
ω^2_{WC}
S_I
S_{TF}, ω_{TO} : Text Source and Word Splitter
ω_{WC} : Word Counter

Figure 24: Example of parallel query structures of trending topic and word count.

Tweets to find trending topic

Trending Topics (TT): The objective of trending topics application [216, 171] is to analyze a continuous stream of tweets and determine the frequency of each topic within a limited window of events as presented in Figure 24. It applies ① `TwitterParser` to parse tweet streams into structured Tweet objects containing essential information like ID, timestamp, and text content followed by using ② `TopicExtractor` (ω_{TE}) to scan the tweet text to extract topics denoted such as " @ " or " # " symbols. ③ `RollingCounter` (ω_{RC}), a sliding window mechanism is then applied to aggregate topic counts over time intervals which are exceeding a predefined popularity threshold ③ `PopularityDetector` (ω_{TR}) are considered trending topics.

Text analysis

Word Count (WC): The Word Count application [144] is a primary example in DSP systems, commonly used to demonstrate real-time text processing capabilities by dividing a continuous text stream into individual words and counting their occurrences as presented in Figure 24. The application's workflow is managed through a series of connected operators: ① `Tokenizer` operator (ω_{TO}) splits the incoming sentences into words, assigning a token, typically the integer 1, to each word, thus transforming the data stream into a series of tuples where each tuple consists of a word and the number 1 followed by ② WordCounter (ω_{WC}) operator receives the tokenized stream and aggregates it by word, summing the tokens associated with each word to calculate the total occurrences of each word.

4.5 PDSP-Bench Controller

Orchestrates seamless benchmarking

In the context of DSP, the heterogeneity and distributed nature of such systems pose significant challenges (**C2**) for benchmarking. The PDSP-Bench's controller is designed to address these challenges by serving as a central management hub, orchestrating the benchmarking process across diverse DSP configurations. It facilitates a seamless benchmarking experience by abstracting the end-to-end complexities such as resource setup and DSP deployment from the user. This separation ensures that the benchmarking process accurately measures the SUT's performance without extraneous overhead. The controller simplifies the benchmarking process through automation, relying on user inputs through the web user interface (cf Section 4.6) to manage distributed resources and collect results.

Benchmarking process coordination

Automation Manager is responsible for initiating and coordinating the various components of the benchmarking process. It automates and triggers the execution of various tasks such as workload generation, deployment of parallel query plans, and collection of benchmark results. It is based on the need for a streamlined, error-free process that eliminates manual intervention and ensures consistency across benchmark runs. For instance, ensuring that each node receives the correct workload and executes the appropriate part of the query plan in a distributed DSP setup is critical for a fair assessment. The *Automation Manager* coordinates these activities, reducing the complexity of managing distributed systems.

Interface to workload generation

Workload Manager manages the retrieval and distribution of workloads, interfacing with both the *Workload Enumerator* and the *message broker*. It ensures that the correct data streams, whether synthetic or derived from real-world applications, are fed into the SUT according to the specified parallel query plans. This is crucial for emulating real-world conditions, where SUT must handle varied dynamic and unpredictable data volumes. Accurately simulating these conditions, PDSP-BENCH can assess how well a SUT scales and manages resources under different parallel stream processing scenarios. An example of its functionality can be observed in benchmarking of a DSP system designed for IoT data processing. The *Workload Manager* would coordinate the delivery of diverse data streams, simulating sensor data from various IoT devices to assess the SUT's throughput and data processing capabilities.

Handles ML pipeline

ML Manager. The inclusion of a ML Manager reflects the evolving nature of DSP benchmarking, where traditional static analysis may not fully capture the dynamic behavior of these systems under various loads and conditions (cf. **C3** Section 4.1). The *ML Manager* relies on *Workload Enumerator* to provide a diverse set of PQP and corresponding historical performance data to train models to predict system performance under different conditions or can be used to optimize the selection of test parameters such as parallel query plans. These predictive capabilities are relevant to future DSP for more intelligent testing and can significantly enhance the benchmarking processing, where the benchmark suite can adapt to the observed performance characteristics of the SUT. For instance, if ML models identify a particular query plan as resource-intensive, subsequent tests can probe this scenario to identify the most effective parallelism strategies for a given workload based on learned patterns from previous tests. Thus, *ML manager* represents a forward-thinking approach to benchmarking future DSP, enabling dynamic adaptation to the evolving behaviors of DSP.

Message Broker. Including a message broker within the PDSP-Bench facilitates the efficient distribution of workloads to the SUT. It acts as an intermediary layer that facilitates the decoupling of data producers from consumers, allowing the *Workload Manager* to publish data streams without direct coupling to the SUT, enabling a more flexible and realistic dynamic test environment. For instance, PDSP-Bench uses Kafka as the message broker to handle high-throughput and distribute a high volume of data streams across distributed clusters, which is critical for the benchmarking of SUT.

Simulate real-world stream data

4.6 Web User Interface

In the context of DSP benchmarking, the complexity of configuring various parallel query plans and managing heterogeneous resources often poses significant challenges to users with limited expertise in parallel processing or benchmarking. To address this challenge led to developing a web user interface (WUI) within PDSP-Bench to simplify the benchmarking process. It serves as a portal through which users can effortlessly deploy homogeneous and heterogeneous clusters, configure DSP settings, select workloads and a variety of parallel query plans derived from both real-world applications and synthetic benchmarks. In addition, PDSP-Bench's WUI offers an intuitive platform for configuring benchmarking parameters, such as parallelism degree, event rates, and window sizes. These parameters are pivotal in assessing the scalability and performance of DSP under varied workloads. By abstracting the complexities of benchmark setup and parallel query execution, the WUI enables users to focus on the core objective of benchmarking—the evaluation of DSP performance against defined metrics and scenarios.

Managing users input...

...for orchestrating benchmarking process.

The WUI also acts as a gateway, forwarding user-defined benchmark configurations to the PDSP-Bench's controller, which orchestrates the initiation of the benchmarking process (cf. Section 4.5). Performance metrics and query outputs are subsequently published to a Kafka output topic, from where they are directed to the *results' visualizer.* This visualization component enables real-time monitoring of key performance indicators such as end-to-end latency and resource utilization, offering immediate insights into the DSP' behavior under test conditions. Moreover, the *controller* archives every test configuration and its resultant performance metrics in a database. This archival facilitates historical analysis and comparisons, enriching the benchmarking process with a temporal dimension. Through this, users can track performance trends over time, compare the efficacy of different configurations, and make informed decisions on optimizing DSP performance. Lastly, by encapsulating the benchmarking process within an intuitive web interface, PDSP-Bench significantly enhances the accessibility and utility of the system. Researchers and professionals can focus on the core objective of benchmarking DSP capabilities, free from the encumbrances of complex setup procedures and configurations. This streamlined approach not only facilitates a more efficient evaluation of DSP across diverse scenarios but also promotes a broader adoption of PDSP-Bench in both academic and industry settings.

4.7 Summary

In this chapter, we propose the first contribution of this thesis PDSP-Bench, a novel performance benchmarking system to tackle the research challenge (cf. RC1 Section 1.1) of systematic understanding of PDSP in DSP systems. The critical analysis of existing benchmarking systems for DSP systems reveals significant gaps, particularly in handling parallel dataflows or PQP and adapting to heterogeneous computing environments. Furthermore, integrating machine learning (ML) into DSP systems presents new opportunities and challenges. While ML can enhance parallelism and resource allocation strategies, existing benchmarks do not adequately support ML model testing and comparison. They cannot often provide consistent metrics or generate diverse data streams required for training ML models, which is crucial for their application in dynamic streaming environments.

To solve these challenges, PDSP-Bench, a comprehensive benchmarking system is designed to address the complexity of benchmarking parallel and distributed stream processing across heterogeneous environments. It offers *(i)* the most extensive suite of benchmarks, including 14 real-world applications and 9 synthetic workloads, encompassing both standard stream processing and user-defined operators. *(ii)* It is designed to rigorously test DSP systems across various hardware configurations, thereby enhancing our understanding of system behaviors under diverse operational conditions. *(iii)* PDSP-Bench also features an intuitive web interface that facilitates the scaling of workload generation for both data streams and queries, enhancing the benchmark's utility in machine learning applications within DSP systems. This capability is critical as it allows researchers and developers to simulate and analyze the performance impacts of different parallelism strategies and hardware configurations. It provides invaluable insights that drive the optimization of DSP. In Section 6.2, we present results from an extensive evaluation that demonstrates the scalability and capabilities of PDSP-Bench to benchmark a well-known DSP Apache Flink using parallel query structures, varying workload, and hardware configuration and highlight observations from our experiments.

5

ZeroTune: Zero-Shot Cost Model for Performance Prediction

Distributed Stream Processing (DSP) is an essential paradigm for real-time data analysis, extensively used across various application domains. Leading organizations like *Alibaba* in retail [153], *King* in online gaming [49], and *Netflix* in video streaming [33] harness the power of DSP to handle core operational tasks. Alibaba, for instance, processes an average of 4 million transactions per second, with demand sometimes peaking even higher. In this scenario, it requires processing enormous data volumes across numerous parallel operator instances to ensure consistent performance and reliability. The diverse and intense demands of such applications pose substantial challenges in parallelism tuning, as tuning the degree of parallelism is critical because it must be aligned with the specific characteristics of the workload and underlying resources. Incorrect provisioning in parallelism can lead to backpressure—a condition where data input rates exceed the processing capacity of the system—leading to significant performance degradation and potential system failures.

Parallelism is used in DSP...

...to process higher workload.

Incorrect provisioning...

However, tuning the parallelism of DSP operators is not straightforward. It requires a nuanced understanding of how configuration changes will affect overall system performance, as any adjustment in parallelism often necessitates substantial changes to DSP query itself, including costly relocations of operators and the splitting of their state to accommodate a new parallelism degree. Therefore, it is crucial to use accurate predictive cost[7] models to provide reliable forecasts of key performance metrics, such as latency and throughput, before any operational changes are applied. These models help ensure that changes made to the system configuration improve performance as intended without causing disruptions or unforeseen negative impacts.

...due to lack of overall performance.

Accurate performance modeling is the need of PDSP

To solve the second research challenge on performance prediction and optimization, we are ZeroTune for accurate performance prediction for determining optimal parallelism without query execution for parallel and distributed stream processing (PDSP). Figure 25 illustrates the system architecture of ZeroTune, focusing on accurate performance prediction and op-

[7] Performance and cost are used interchangeably in this thesis.

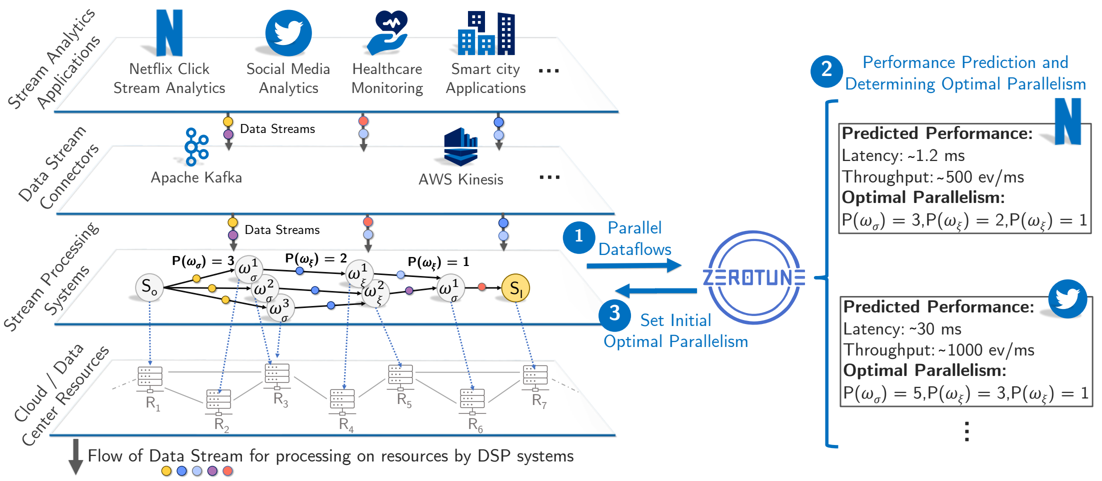


Figure 25: Overview of ZeroTune architecture for performance prediction and determining initial parallelism tuning of parallel query structures for execution via DSP systems across various resource configurations and under different workload conditions stemming from a diverse range of applications.

Cumbersome manual parallelism tuning process

timizing the initial parallelism degree of each operator. This architecture is built on the foundational concepts, as introduced in the previous chapter (cf. PDSP-Bench Chapter 4), understanding and performance prediction of parallel data flow in distributed and heterogeneous environments.

Traditional DSP systems often rely on manual tuning to determine the optimal parallelism, a process that proves particularly cumbersome within the distributed and heterogeneous environments typical of DSP. This complexity is due to the variation in workloads, including differences in query plans and data streams, alongside the diverse array of resource configurations. Identifying the optimal parallelism manually involves executing numerous configurations to empirically determine the best configuration, a method that is both time-consuming and inefficient.

Several oscillations before optimal parallelism

Additionally, while heuristic and analytical methods exist [108, 83, 131, 174], these non-learned (cf. Section 2.2.2) approaches also typically require multiple adjustments, or oscillations, before settling on an optimal parallelism. Learned or machine learning (ML) approaches [201, 264, 54], although potentially more adaptive, often suffer from similar limitations (cf. Section 2.2.2), requiring several iterations to fine-tune the model and typically being tailored to specific workloads without the ability to generalize effectively across different scenarios without a high amount of data and training time.

Costly retraining for every new workload

In contrast, ZeroTune is designed to overcome these challenges by predicting accurate performance, e.g., latency and throughput and using the predicted cost to determine optimal initial parallelism degree. This approach leverages characteristics of the workload and resources to make informed predictions about performance and determine parallelism without the need to execute the query.

5.1 Analysis of Performance Prediction Methods

Designing an accurate cost model for operator parallelization remains a critical challenge (cf. *RC2* Section 1.1). Many current learned-based solutions for performance prediction and determining parallelism [108, 201, 264, 192, 99, 54, 217], employ online learning to dynamically predict parallelism degrees by monitoring query performance in real-time. However, existing approaches introduce several significant challenges for DSP systems in performance prediction and determining parallelism:

C1: Incorrect Initial Provisioning: Although online learning may seem advantageous for scaling decisions within stream processing, it often leads to highly inaccurate initial provisioning for parallel operators. This necessitates multiple adjustments before achieving a stable state, resulting in prolonged convergence times [131]. Such iterative adjustments to achieve the desired parallelism degree are impractical for real-time applications, like online gaming, where they can cause significant delays.

Cumbersome manual parallelism tuning process

C2: Non-transferable Features: Online learning models are typically trained on context-specific features, referred to as *non-transferable features*. These features may be effective within specific scenarios—such as a temperature filter literal of "≤ 27 degrees" may be useful for specific data streams (e.g., weather reports) but fail to generalize across different contexts, like smoke detection systems [201]. A more robust approach would involve learning from context-independent features, such as the complexity of filters or the number of attributes used, which can provide better generalization across various streaming environments.

ML models are workload dependent

C3: High Training Effort: While some models achieve a degree of generalization, they require substantial training efforts, which is not feasible for all applications. For example, established models within the database field need extensive training on over $200k$ queries and 15 different databases to generalize effectively across unseen databases [114]. Unlike database systems, DSP workloads and data characteristics are often unpredictable, necessitating models that can handle a wide variety of workloads and effectively reduce the training burden for managing unseen or new workload types.

Hard to achieve generalization for DSP

In this chapter, we introduce ZeroTune, a model designed to initially configure parallelism degrees based on accurate performance prediction of parallel query plans. This approach helps to avoid costly trial-and-error adjustments from the start of query execution. Although ZEROTUNE can also adapt parallelism degrees during runtime, our primary focus is demonstrating its capability to make well-informed initial configurations. ZEROTUNE employs a novel learning paradigm that leverages recent advancements in transfer learning, particularly in data-efficient zero-shot learning [46, 222]. This method enables ZEROTUNE to grasp the dynamics of a DSP system offline, allowing it to apply learned insights across various DSP queries and applications without the need for costly retraining. The cornerstone of ZEROTUNE is the use of *transferable* features attributes that maintain consistent semantic significance across different workloads. For example, ZEROTUNE uses the filter operator "$\leq$" to learn about the cost implications (latency and throughput) that are independent of specific data streams.

ZEROTUNE *for performance prediction*

As mentioned before, the main objective of ZEROTUNE is to overcome challenges by achieving generalizability and precise cost prediction while minimizing the training dataset required for effective training. This approach ensures that ZEROTUNE can provide reliable and efficient initial tuning for parallelism, significantly reducing the operational overhead and enhancing the performance of DSP systems.

We have developed a novel method for predicting costs for PQP using a GNN architecture (cf. Section 2.2.2) for learning and encoding transferable features. In this model, each node within the graph represents a crucial component of the parallel DSP system, embodying transferable features such as parallel operator instances, partitioned data streams, and their distribution across various hardware resources. These nodes are characterized by distinctive attributes and their connections with other nodes in the system, enabling the model to decipher the intricate relationships and dependencies within the PQP.

Transferable features...

By utilizing this graph-based technique, the proposed model, ZEROTUNE, achieves the capability to adjust to new, previously unseen parallel operators and hardware setups. This flexibility significantly broadens ZEROTUNE's range of applications, enhancing its effectiveness in optimizing parallel query executions across varied computational environments. Such adaptability ensures that ZEROTUNE remains applicable and effective, even as system configurations and requirements evolve.

...and graph representation...

ZEROTUNE significantly outperforms models that utilize non-transferable features by delivering highly accurate cost predictions, as demonstrated in Figure 26. Notably, for unseen query types, it surpasses existing learned cost models by a factor of 1000, particularly those employing a flat vector representation, in favor of our advanced graph-based approach. Additionally, ZEROTUNE benefits from an innovative data-efficient training strategy, OptiSample,

...are the keys for prediction accuracy and generalization.

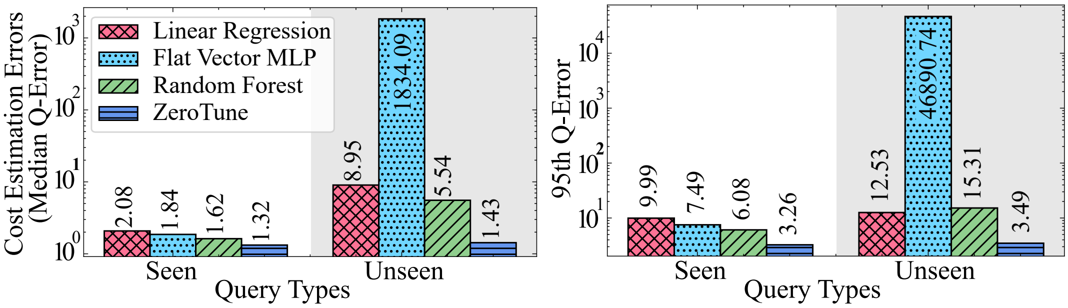


Figure 26: Comparison of ZeroTune with non-transferable cost prediction models [8]. ZeroTune excels in providing robust and generalized predictions of latency costs for both seen and, notably, unseen parallel query plans. This performance stands in stark contrast to other models that rely on *non-transferable* features [90, 45, 13], which typically struggle with generalization across different configurations.

which leverages analytical methods [83, 131] to explore parallelism degrees. This approach enables ZeroTune to generalize from as few as $5k$ queries, significantly reducing the need for the data compared to traditional models, which require $4\times$ more data to achieve similar results. These advancements are further detailed in our subsequent evaluations.

5.2 ZeroTune Overview

The primary objective of ZeroTune is to predict the performance or costs, such as latency and throughput, for executing parallel stream processing queries on distributed and parallel hardware resources, specifically for queries and configurations that have not been previously observed, referred to as *unseen*. Below, we outline our solution (S) to address these challenges (C) effectively.

C1: Incorrect initial parallelism. To ensure accurate initial provisioning of operator parallelism at the deployment stage, we utilize an offline supervised model as described next.

S1: Figure 27 presents the overview of ZeroTune. It is initially trained using a diverse set of DSP workloads such as data streams, parallel query structures (PQP), and heterogeneous configurations of resources by identifying a set of transferable features (refer C2). Once trained, ZeroTune is capable of accurately predicting costs (latency and throughput) for unseen queries and resource combinations. An optimizer subsequently utilizes these predictions to determine the most cost-effective parallelism configuration. Unlike previous approaches, ZeroTune establishes optimal parallelism degrees from the onset and is adaptable to unseen workloads.

Different DSP workloads to train ZeroTune

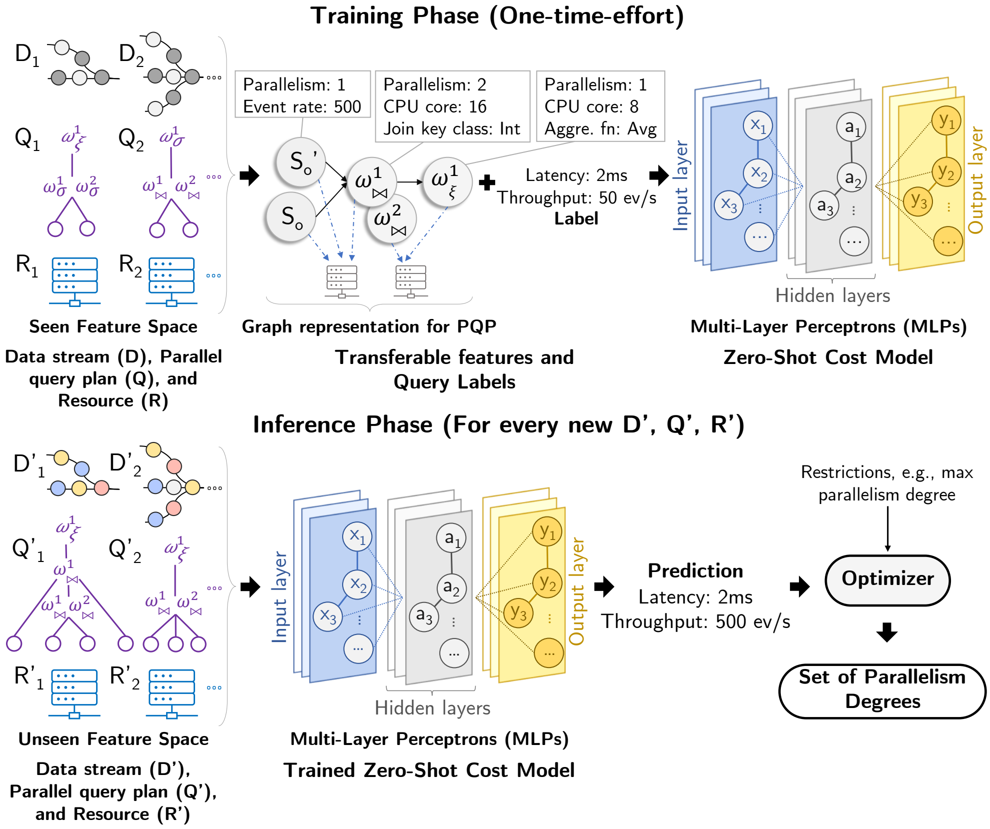

Training Phase (One-time-effort)
D1 D2
Q1 Q2
R1 R2
Parallelism: 1 Event rate: 500
Parallelism: 2 CPU core: 16 Join key class: Int
Parallelism: 1 CPU core: 8 Aggre. fn: Avg
S_o'
S_o
Latency: 2ms
Throughput: 50 ev/s
Label
Input layer
Hidden layers
Output layer
Seen Feature Space
Data stream (D), Parallel query plan (Q), and Resource (R)
Graph representation for PQP
Transferable features and Query Labels
Multi-Layer Perceptrons (MLPs)
Zero-Shot Cost Model
Inference Phase (For every new D', Q', R')
D'1 D'2
Q'1 Q'2
R'1 R'2
Input layer
Hidden layers
Output layer
Prediction
Latency: 2ms
Throughput: 500 ev/s
Restrictions, e.g., max parallelism degree
Optimizer
Set of Parallelism Degrees
Unseen Feature Space
Data stream (D'), Parallel query plan (Q'), and Resource (R')
Multi-Layer Perceptrons (MLPs)
Trained Zero-Shot Cost Model

Figure 27: Overview of ZEROTUNE for performance or cost prediction applied to parallelism tuning [8]. We initially train ZEROTUNE (top figure) using transferable features from a variety of streaming workloads with varying parallelism levels. Once trained, the model is utilized during *the inference phase* (bottom figure) to accurately predict the cost of unseen parallelism query structures (PQP) without needing expensive retraining. The predicted cost from ZEROTUNE is used by an *optimizer* to determine parallelism degrees of PQP that minimizes the overall execution cost.

C2: Generalizability. A crucial aspect of ZEROTUNE is its ability to generalize across diverse workloads involving parallel and concurrent operators on various hardware setups. This is inspired by established practices in creating generalizable cost models for databases [113, 114, 6, 5]. Our approach is two-fold: (1) employing a novel feature selection process that focuses on *transferable features*, and (2) utilizing a generic graph representation to model the query plan and learn from operational characteristics. The challenge is to understand the dependencies associated with parallel resource allocation and the nuances of applying this knowledge to unseen PQP on multicore hardware.

Encoded transferable features...

S2: Transferable features. We focus on learning from attributes such as the *data-partitioning* schemes of an operator and the *CPU cores* used by the hardware resources. These features help ZEROTUNE understand the impact of load distribution on overall costs. Such transferable features describe the operational aspects of query operators and their deployment on hardware, enabling the model to discern relationships and transfer knowledge from known (*seen*) to unknown (*unseen*) scenarios.

...with parallel graph representation...

S3: Parallel graph representation. Our second major contribution is the innovative use of parallel graph representation. This format naturally depicts both operators and their interconnections, with "nodes" representing operators endowed with transferable features and "edges" illustrating data flow among them. This structure, represented as a DAG suitable for GNN applications, includes operator-instance resource mapping as additional edges, enhancing the model's predictive accuracy. As data circulates through this graph, a message-passing mechanism updates the hidden states of nodes, culminating in a comprehensive state that informs the final cost estimation regression. This robust graph-based method, combined with strategic feature selection, ensures precise cost predictions for new PQP configurations during the inference phase.

...to enable generalization by transferring knowledge.

C3: Enormous training effort. Traditional models require training on a vast array of queries. For instance, $200k$, queries across as many as 15 different databases [114] for achieving generalization for cardinality examination. Gathering such a diverse set of data for stream processing presents significant challenges due to the inherently dynamic nature of data streams, which continuously evolve. Additionally, the scarcity of comprehensive benchmarks for stream processing compounds the difficulty of assembling sufficient data to train large models effectively.

Data-centric approach...

S4: Data-efficient Training. Given the complexities of acquiring diverse training datasets—encompassing various data streams, parallel query plans (PQP), and hardware configurations—it is often impractical and costly to undertake such extensive data collection in the stream processing context. To address this, we introduce a hybrid training approach, OptiSample, that strategically varies data streams, query parameters such as window size, and resource configurations from basic to high-end setups. However, it restricts the exploration to specific parallelism degrees of PQP as suggested by existing research [131, 83]. This method aims to cultivate a generalizable model that requires less data and reduces training time, details of which are further discussed in Section 5.5.

...for efficient data generation and training.

5.3 Zero-shot Cost Model

ZeroTune is designed to predict the costs associated with executing PQP that have not been previously observed, i.e., *unseen* on specific combinations of resources. The approach leverages a set of transferable features to estimate cost metrics effectively without direct observation of PQP deployment, as detailed below, following the definitions of these metrics.

5.3.1 *Metrics for Prediction Model*

Network latency to consider...

...delay between operator instances.

The objective of ZeroTune is to predict cost metrics that can be used to optimize parallelism degrees across physical resources, aiming to minimize the total execution cost. The model primarily focuses on two critical cost metrics commonly used in streaming applications: end-to-end latency (cf. Definition 6) and throughput (cf. Definition 7). These metrics have been long-established by Stonebraker et al. [221] and remain relevant in contemporary applications [133, 164, 42]. While these are the primary metrics, ZeroTune is adaptable and can be calibrated to assess additional metrics such as resource utilization and availability [53], simply by adjusting the final Multi-Layer Perceptron (MLP) node in the model.

5.3.2 *Transferable Featurization*

Transferable features of...

...query, operator and resources.

We explore the set of transferable features within the context of PQP cost prediction. Transferable features are defined as attributes associated with the query, data stream, and hardware that, once learned, can be effectively utilized to predict costs for scenarios involving *unseen* queries, data streams, and hardware configurations. These features are crucial because they allow the predictive model to generalize across different operational contexts without direct prior exposure. We categorize transferable features into three primary groups to streamline their application in diverse scenarios. Each category targets specific aspects of the streaming process and system architecture, enhancing the model's ability to adapt its predictions to new environments. The details and rationale behind selecting these features are listed in Table 5 and thoroughly discussed in the subsequent sections.

Operator Parallelism-related Features

Transferable feature selection study...

In our feature selection study, we focus on identifying and understanding the features critical to cost prediction for PQP within DSP systems. Among these features, the *parallelism degree* of operators stands out as a pivotal factor that significantly impacts the execution costs. To explore the effects of parallelism degree on key performance metrics—namely latency and throughput—we carry out a detailed micro-benchmark, the results of which are pre-

Node	Category	Feature	Description
Logical	operator-parallelism	Parallelism degree	Parallel instances of the operator
	operator-parallelism	Partitioning strategy	Strategy for data distribution (forward, rebalance, hashing)
	operator-parallelism	Grouping number	Number of operators grouped together by DSP
	data	Tuple width in	Input tuple width
	data	Tuple width out	Output tuple width
	data	Tuple data type	Data types in a single tuple
	data	Selectivity	Average selectivity of all instances of a given operator
	data	Event rate	Emitted event rate of the source
	operator	Operator type	Type of operator
	operator	Filter function	Comparison filter function, e.g., $<, \leq, \geq$
	operator	Filter literal class	Filter literal datatype for comparison, e.g., int
	operator	Window type	Shifting strategy (tumbling/sliding)
	operator	Window policy	Windowing strategy (count/time)
	operator	Window length	Size of the window
	operator	Sliding length	Size of the sliding interval
	operator	Join key class	Join key data type
	operator	Agg. class	Aggregation data type
	operator	Agg. function	Aggregation function, e.g., *min, max, avg*
	operator	Agg. key class	Aggregation key data type
Physical	resource	CPU cores	Number of processing cores
	resource	CPU frequency	CPU frequency on this instance
	resource	Node identifier	Unique identifier of every instance
	resource	Total memory	Available memory of the node
	resource	Network speed	Network link speed between nodes

Table 5: Transferable features to derive costs of PQP categorized into *logical* and *physical* node. A logical node consists of features related to operators, data streams, and parallelism, while the physical node represents features related to resources and their characteristics [8].

sented in Figure 28. The experiment is conducted using *Apache Flink* on a *count-based tumbling window* query. During this test, all variables except for the parallelism degree are held constant to isolate its impact. The setup is designed to maximize cluster utilization while avoiding backpressure, even as parallelism increases. Our observations show a clear trend: as the parallelism degree increases, leading to a decrease in latency and an improvement in throughput. This indicates that higher parallelism levels can efficiently distribute workload across the system, thus enhancing overall performance. However, our analysis also uncovers a critical *non-linearity* in the relationship between parallelism degree and performance metrics at higher levels of parallelism. This non-linearity is particularly influenced by *operator chaining*, a feature we highlighted in our results. Operator chaining in DSP systems, such as Flink, allows multiple independent operators sharing the same parallelism degree to be grouped and executed on a single computational unit or physical node. This arrangement minimizes inter-process communication overhead and maximizes the utilization of available processing cores. The sig-

...to study the impact on cost.

Increasing parallelism degree...

...improves cost.

Operator chaining...

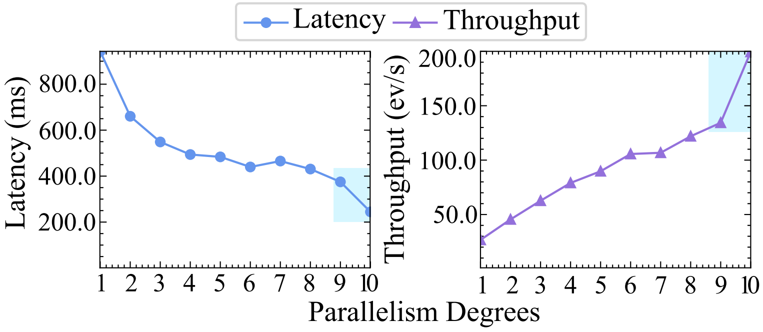


Figure 28: Analyzing the influence of selected features on cost metrics during feature selection study [8]. For instance, impact of increasing parallelism degree and operator chaining or grouping on cost metrics: latency (left) and throughput (right) for a linear query. Notably, decrease in latency and increase in throughput with increasing parallelism degree. Particularly at higher parallelism levels, the strategic grouping of operators—visually highlighted in blue—demonstrates a marked improvement in cost efficiency. This grouping reduces inter-node communication and enhances resource utilization, leading to significant enhancements in performance metrics.

...accelerate performance...

nificant reductions in latency and improvements in throughput observed due to operator chaining underscore its importance. Consequently, our feature selection for the cost prediction model considers not only the parallelism degree of individual operators but also the grouping number, which reflects how many operators are clustered together on nodes during query execution. This dual consideration helps in creating a more accurate and robust prediction model, effectively capturing the dynamics of parallel execution within DSP systems.

...by placing operators together with the same parallelism.

Additionally, the cost implications of a parallel query plan are significantly influenced by the *partitioning schemes* utilized by operators, which determine how workload is distributed across various parallel instances. The selected partitioning strategy plays a crucial role in defining these cost dynamics. In the proposed system ZeroTune, we incorporate several partitioning approaches, including *forward*, *rebalance*, and *hashing* schemes [52]. Each of these strategies offers distinct advantages for load distribution: *forward* passes data to the next operator without alteration, *rebalance* distributes data evenly across all operators, and *hashing* ensures that data with the same key is processed by the same processor, optimizing load handling and enhancing overall system efficiency as explained in Section 3.2.

Operator-related Features

Beyond parallelism-centric features, other *transferable* attributes related to data streams and operators are also critical due to their significant impact on the runtime costs of PQP. For instance, for window operators within a query (as depicted in step ⓪ in Figure 29), we consider factors such as the type of window (tumbling versus sliding), the window policy (time-based or count-based), and the dimensions of the window like window length and the sliding interval for sliding windows. Similarly, for aggregation operators, essential features include the aggregation function employed and the data type of the aggregate. For filter operators, we examine the filter function, the data type of the filter literal, and the average selectivity, which influences how much data passes through the filter.

Event rate to learn from...

...different rate of data streams.

Additionally, understanding the performance impact of data stream-related features is crucial in DSP systems where data characteristics are inherently dynamic and substantially different from those in traditional databases. In contrast to databases, where data distributions are predetermined, DSP systems handle streaming data where prior knowledge of data characteristics is limited. To effectively manage operator parallelism in such environments, it's important to focus on generalized data stream attributes like the *event rate* (data arrival rate) and *tuple width* (the size of data elements). These attributes directly influence the processing demands placed on operators, thereby affecting the required level of parallelism.

No prior knowledge of...

...underlying data distribution in DSP.

Another vital feature related to data streams is the selectivity of operators, which describes the fraction of incoming data that satisfies certain conditions within the filter operators. We consider the average selectivity of the parallel instances of each operator as selectivity can differ significantly due to diverse deployments across physical nodes, impacting performance outcomes. For example, in a scenario where a filter-aggregate query processes two distinct data streams with identical input rates but differing data characteristics, the selectivity feature becomes indispensable. Without incorporating selectivity into the model, a cost prediction system might fail to accurately assess the computational demands for the aggregate operation on each stream. This consideration mirrors strategies employed in database management systems (DBMS) to optimize query performance [114], which shows the importance of context-sensitive cost modeling in dynamic data environments.

Selectivity of operators...

...to learn computational demands.

Resource-related Features

An integral component of cost estimation for PQP involves considering the underlying parallel resources utilized for deployment. The characteristics of

Resource features to capture...

the computing hardware are pivotal for maximizing the efficiency of operator parallelization and enhancing the performance during the physical execution of queries. Recognizing and exploiting specific hardware attributes associated with parallelism allows for the optimization of DSP applications across diverse hardware setups.

...correlation between resources...

...and mapping with operators.

We focus on identifying and incorporating fixed hardware characteristics that are crucial for parallelization. Notable among these are the *number of processing cores and CPU speed*, which serve as representative *transferable* features for different types of hardware. These features are directly linked to the achievable level of parallelism and, consequently, the overall cost in terms of latency and throughput. For example, the number of available processing cores within a physical node fundamentally influences the extent of parallelism that can be realized for an operator. A higher count of processing cores typically enables a greater degree of parallelism, which in turn facilitates quicker computational times, reduced processing latency, and enhanced throughput.

In ZeroTune, we employ distributed physical nodes featuring varied hardware configurations to execute and gather training data for parallel query plans, allowing to systematically explore different degrees of parallelism and their impact on performance metrics. Additionally, we encode the mappings among logical operators, physical nodes, and the associations from logical operators to physical nodes. This graphical representation helps delineate the hardware characteristics and their effects on operator parallelization and overall performance. Through this detailed approach, we aim to ascertain the most effective strategies for parallelizing operators across a spectrum of hardware configurations, thereby optimizing resource utilization and achieving superior performance outcomes.

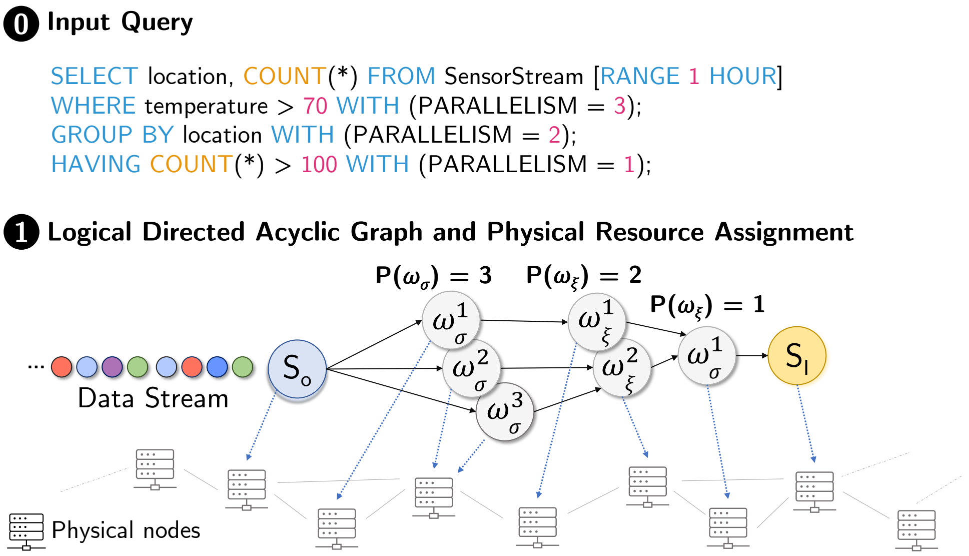
5.4 Training and Optimization

ZeroTune *is trained only once...*

In (Figures 29 to 32), we outline the model architecture of ZeroTune, detailing the training and inference process, including our novel parallel graph representation, and demonstrating how an optimizer leverages cost predictions for effective parallelism tuning.

5.4.1 *Training and Inference*

The ZeroTune model is trained in an offline supervised manner, where the queries are visually represented in a graph format. The graph representation



❶ **Input Query**

```
SELECT location, COUNT(*) FROM SensorStream [RANGE 1 HOUR]
WHERE temperature > 70 WITH (PARALLELISM = 3);
GROUP BY location WITH (PARALLELISM = 2);
HAVING COUNT(*) > 100 WITH (PARALLELISM = 1);
```

❶ **Logical Directed Acyclic Graph and Physical Resource Assignment**

Figure 29: A detailed view of ZeroTune model architecture showing logical and physical execution plan of an arbitrary PQP. Step ⓪ Logical plan represents the flow of data stream between operators of PQP. Step ① ZeroTune captures the physical execution plan i.e., the mapping of these operators and its instances on diverse and distributed resources for PQP execution. Different parallelism enumeration strategies can be applied to set parallelism of operators and deployment on physical resources [8].

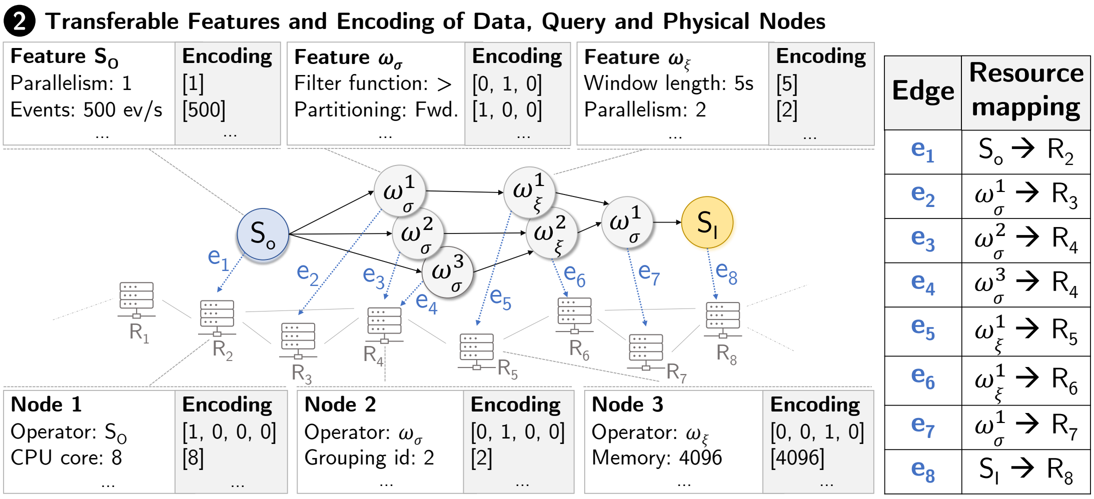
incorporates encoding of transferable features across both logical and physical nodes using GNN model (cf. steps ⓪ and ① of Figure 29). In this model, each graph node is functionally akin to a multi-layer perceptron (MLP), embedding the physical and logical nodes into the input and hidden layers systematically (as outlined in step ② of Figure 30). This structured embedding underpins the foundational graph representation devised for the PQP.

...using encoded transferable features.

The process of neural message passing initiates between the operator nodes (illustrated with black edges), continues between physical nodes (orange edges), and culminates with operator-resource mapping (green edges). This sequence facilitates the aggregation of information at the sink, where predictions are finalized (cf. step ③ of Figure 31). For instance, in the provided figure, the hidden state of the node representing the first instance of a filter operator (ω^1_σ) is derived by inputting the feature vector x_1 (which includes transferable features from the data source) into MLP shared across all upstream nodes (gray nodes).

Message passing includes resources nodes

The sequence of message passing integrates the input and output of the hidden states along both the data flow and physical operator mapping as shown in step ③. This integration is crucial for the model to accurately learn the runtime behaviors of each parallel operator instance across different hardware resources and, conversely, to pinpoint precise costs associated with them. Af-



❷ Transferable Features and Encoding of Data, Query and Physical Nodes

Feature S_O	Encoding
Parallelism: 1	[1]
Events: 500 ev/s	[500]
...	...

Feature ω_σ	Encoding
Filter function: >	[0, 1, 0]
Partitioning: Fwd.	[1, 0, 0]
...	...

Feature ω_ξ	Encoding
Window length: 5s	[5]
Parallelism: 2	[2]
...	...

Node 1	Encoding
Operator: S_O	[1, 0, 0, 0]
CPU core: 8	[8]
...	...

Node 2	Encoding
Operator: ω_σ	[0, 1, 0, 0]
Grouping id: 2	[2]
...	...

Node 3	Encoding
Operator: ω_ξ	[0, 0, 1, 0]
Memory: 4096	[4096]
...	...

Edge	Resource mapping
e_1	$S_o \rightarrow R_2$
e_2	$\omega_\sigma^1 \rightarrow R_3$
e_3	$\omega_\sigma^2 \rightarrow R_4$
e_4	$\omega_\sigma^3 \rightarrow R_4$
e_5	$\omega_\xi^1 \rightarrow R_5$
e_6	$\omega_\xi^1 \rightarrow R_6$
e_7	$\omega_\sigma^1 \rightarrow R_7$
e_8	$S_I \rightarrow R_8$

Figure 30: Parallel graph encoding or representation of PQP within ZeroTune model architecture. Step ② A detailed view shows transferable features associated with workload (parallel data streams and PQP) and different resources based on allocation of these operators in resources. In ZeroTune, parallel graph representation captures these intrinsic correlations as graph representation with encoded transferable features of different operators and resources as graph node types [8].

ter the message passing, the final node output (y_1) provides predictions for metrics such as latency or throughput during the inference phase (cf. step ④ of Figure 32). This comprehensive training and inference mechanism enables ZeroTune to effectively predict the costs of executing unseen PQP on unobserved resource combinations, ensuring optimal parallelism configuration from the outset.

5.4.2 *Parallel Graph Encoding*

Redundant feature information...

A major challenge in parallel graph encoding or parallel graph representation is encoding the attribute space of PQP effectively within the graph, while avoiding increased model complexity or degrading the performance of the cost models. To address this, we evaluated two potential strategies for incorporating the defined transferable features into our graph representation: (1) representing each operator instance and its features as an individual graph node, or (2) encoding only distinct operators separately while grouping parallel operator instances along with parallelism-related features into a single node.

...in individual node encoding.

Individual node encoding: The first approach (1) tends to introduce redundancy by replicating similar or identical feature information across multiple graph nodes, which corresponds to each parallel operator instance. This redundancy not only inflates the graph size but also complicates the model

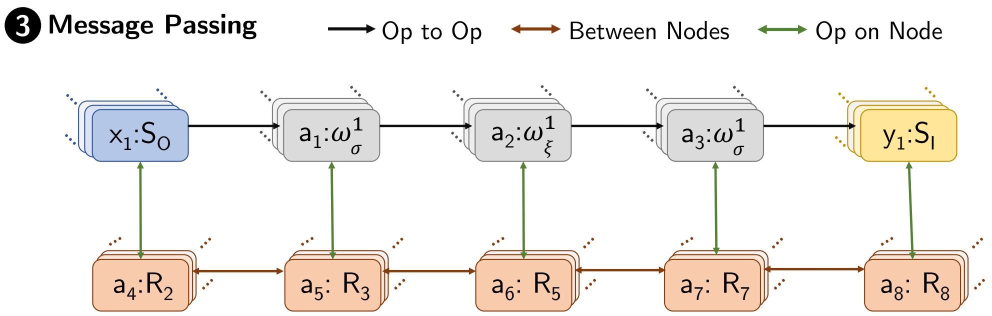


Figure 31: Message passing among graph node for training within ZEROTUNE model architecture. Step ③ A detailed view on the message passing for training where each operator's feature vectors, along with its attributes, are encoded into a node-specific MLP to generate a hidden state. This hidden state is then propagated through the graph in the specified order of the query, ultimately reaching the sink [8].

unnecessarily, as each parallel operator instance might share extensive commonalities in feature sets. For instance, if we consider a scenario where the maximum parallelism on a machine is $max_P = 64$, a linear query would generate 64 instances of a filter operator, each linked by 64 edges from a single source ($\omega^P_{So} = 1$). This setup would further extend to 128 edges connecting each filter to a window aggregation operator ($\omega^P_\xi = 2$), culminating in an overwhelming 4096 edges for neural message passing. Such a model becomes highly complex without offering substantial new information per operator instance.

Aggregate operator encoding: In contrast, the second approach (2) simplifies the graph structure by aggregating similar operator instances into a single node enriched with averaged or collective attributes of those instances. This method not only reduces the number of nodes and edges in the graph but also focuses on the distinctiveness of each operator type without unnecessary duplication of node information. For example, when averaged across instances, attributes like selectivity provide a more manageable and equally informative feature set for the model. A microbenchmark validating this strategy demonstrated minimal variation in individual selectivities across different partitioning schemes, supporting our aggregation choice.

Aggregated encoding to...

...reduce graph complexity.

Additionally, encoding transferable features of multiple instances into a single graph node introduces the challenge of adequately representing resource mappings for each operator instance. Although we aggregate attributes like selectivity, we meticulously maintain distinct edge information for each parallel operator instance (e.g., ω^i_σ, here i=op. instance). This structure, detailed in the depiction of operator-resource mapping in Figure 30 step ②, allows the model to accurately assess the cost implications associated with each resource and operator instance during inference. By adopting this method, we

Trained models...

...for cost prediction of unseen PQP.

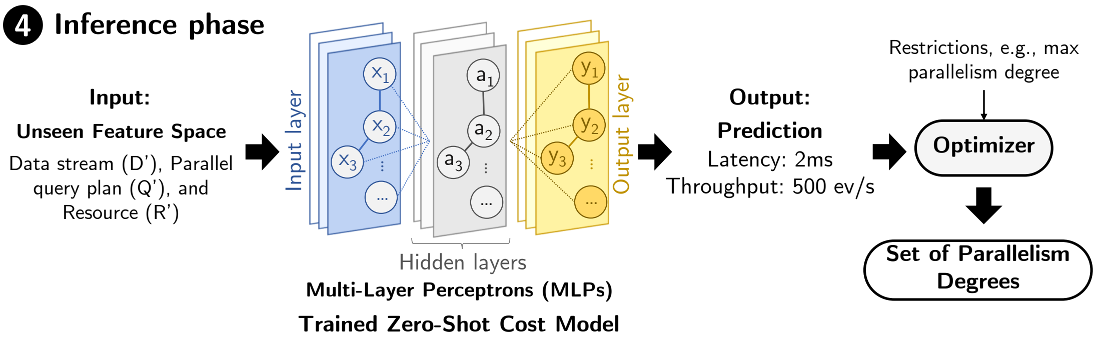

❹ Inference phase
Input:
Unseen Feature Space
Data stream (D'), Parallel query plan (Q'), and Resource (R')
Input layer
x1
x2
x3
...
a1
a2
a3
...
y1
y2
y3
...
Output layer
Hidden layers
Multi-Layer Perceptrons (MLPs)
Trained Zero-Shot Cost Model
Output:
Prediction
Latency: 2ms
Throughput: 500 ev/s
Restrictions, e.g., max parallelism degree
Optimizer
Set of Parallelism Degrees

Figure 32: Performance of cost prediction using ZeroTune during inference phase. Step ④ Final layer MLP provides the predicted cost for an arbitrary PQP. The predicted cost is used by the optimizer to apply *what-if* costs conditions to the PQP to determine the optimal set of parallelism degrees for each operator leading to the minimal overall cost (cf. Equation (12)) [8].

optimize the model's complexity while preserving its ability to reason effectively about diverse resource allocations and their impact on execution costs.

5.4.3 *Optimizer*

Predicted cost used by optimizer...

The ZeroTune cost model is designed to work together with an optimizer, enhancing the decision-making process to minimize the overall costs associated with query execution, as illustrated in Figure 32. For ZeroTune to effectively determine optimal parallelism degrees, it first needs to enumerate potential PQP. However, given the NP-hard nature of this task, an exhaustive enumeration of all possible plans is impractical. To circumvent this limitation, we employ a pragmatic approach where a subset of PQP is randomly selected, and their associated costs are predicted using the zero-shot model. The optimizer then evaluates these plans to identify the configuration that offers the minimal combined cost of latency and throughput. This process is detailed in the subsequent sections, where we formalize the optimizer's role within a combinatorial optimization framework, focusing on achieving the most cost-effective parallelism degrees. The primary objective of ZeroTune is to ascertain a set of parallelism degrees P_i for each operator $\omega_i \in \Omega$ in PQP that minimizes the inferred costs C of executing the query on a specified resource set R. To facilitate this, we introduce a novel OptiSample strategy, elaborated in Section 5.5, which guides the enumeration of feasible parallelism degrees.

...to determine parallelism of operators...

We construct the objective function for the parallelism tuning problem by integrating two pivotal cost metrics: latency (C_L) and throughput (C_T). Both metrics are normalized within the range $[0, 1]$, where 0 signifies optimal performance and 1 represents the least desirable outcome. In this normalization process, throughput is negated to align with the maximization goal, whereas

...that minimizes overall cost.

latency is minimized directly. The relative significance of these metrics is controlled by the weight factor wt, which balances the importance between latency and throughput in the cost calculation:

$$C = \underset{C_L, C_T}{\arg\min}[wt \cdot C_L + (1 - wt) \cdot C_T] \tag{12}$$

s.t. for each operator ω_i, the parallelism degree P_i must be a positive integer, $P_i \in \mathbb{Z}$ and at least 1, $P_i \geq 1$. Additionally, the maximum parallelism degree P_{max_p} is constrained by the aggregate number of processing cores n_{core} across all designated resources R, ensuring that $P_{max_p} \leq n_{core}$.

5.5 Data-efficient Training

To ensure that the ZeroTune cost model effectively generalizes across diverse *unseen* workloads and hardware configurations, we train it using a wide array of workload characteristics and physical resource data. This approach addresses a critical concern: how to appropriately enumerate different PQP and their resource assignments to guarantee comprehensive model training. Although prior research [114] suggests that random sampling across a range of scenarios can aid generalization, merely randomizing parallelism degrees could generate suboptimal and noisy query plans, potentially degrading model performance. For example, setting lower parallelism degrees for upstream operators and higher for downstream ones within a query graph can create backpressure, negatively impacting overall performance.

Impossible to capture all possible combinations

Exhaustive approach results in redundant training data

Furthermore, exhaustive enumeration for even a straightforward query structure, such as a 2-way join, becomes infeasible given the extensive possible combinations. Such an exhaustive approach is not just impractical—it often does not yield meaningful insights. Consider a scenario where N physical resources for placing a 2-way join with 9 operators at varying parallelism degrees $P = p_1, \ldots, p_{maxP}$ could yield $N \cdot |p^{|\omega|}_{maxP}|$ combinations. With $N = 1$ and parallelism degrees ranging from 1 to 20, the total combinations would soar to over 512 billion. This scale of enumeration is not only impractical but also computationally prohibitive.

Data-efficient hybrid approach...

To overcome these obstacles, we introduce OptiSample, a hybrid data collection method that balances exploratory diversity, covering a broad spectrum of parallel query workloads and resource configurations, and the analytical precision required to fine-tune parallelism degrees. This method enables the generation of a manageable and representative subset of potential configurations, reducing the computational overhead and increasing the feasibility of training the model on practical scenarios.

OptiSample not only mitigates the challenges associated with vast enumeration but also ensures that the sampled configurations are indicative of real-world scenarios, thus enhancing the model's ability to generalize to new, unseen conditions. This optimized approach is critical for extending ZEROTUNE's applicability and robustness, making it a versatile tool capable of adapting to varied operational environments and supporting efficient resource utilization in DSP systems. While OptiSample provides a robust framework for enumeration, ZEROTUNE's architecture allows integration with other data-efficient sampling techniques that could further refine and enhance the model's generalization capabilities across different streaming applications.

...to collect and explore meaningful data.

Random Enumeration (ZT-Random). A naive way of enumeration that is adopted by existing learned methods [225], is a random enumeration. Random enumeration strategy enumerates based on the number of nodes and their processing capability, i.e., number of cores. For example, four nodes with eight cores may result in enumerating parallelism from minimum one to maximum *number of nodes x number of cores*. Random strategy is simplified, but like exhaustive strategy, randomly assigning parallelism degrees for each operator can result in *noisy* PQP, which are not beneficial learned cost models. For instance, a random strategy may set higher parallelism for downstream operators. However, setting higher parallelism for the downstream operator may not be beneficial in all scenarios, e.g., join, as leading to backpressure issues and significantly hinders query execution performance. Moreover, randomly selecting parallelism degrees in a pattern such as low-high-low may not provide meaningful insights for the model to learn about the performance.

Noisy plans due to random parallelism assignment

OptiSample Training Strategy. The OptiSample strategy represents a structured approach to determining the appropriate number of processing instances, or parallelism degrees P, for each operator within the operator graph G. This method stands in contrast to random sampling strategies, offering a more systematic means of collecting realistic training data that reflects actual system demands. To effectively implement this strategy, OptiSample leverages key characteristics of both the workload (including queries and data streams) and the physical resources involved.

Limit the exploration of parallelism...

Key parameters such as the input event rate (In_{ER}), operator selectivity (sel), output rate (Out_{ER}), and the number of processing cores available (n$_{core}$) are critical in determining the parallelism degrees for operators throughout the operator graph, from upstream (ω_i) to downstream (ω_j) components. The guiding principle behind this approach is that higher input rates and higher selectivity necessitate increased parallelism. This is because more significant computational resources are required to process the incoming rate of events

...by considering workload characteristics.

efficiently, thereby improving the system's ability to manage backpressure effectively.

This method draws inspiration from the operational logic of existing data-parallel DSP systems [83, 131]. These systems employ scaling controllers that adjust parallelism degrees based on similar parameters to maintain optimal performance under varying loads. By incorporating such dynamic scaling principles, the OptiSample strategy enhances the ability of ZeroTune to simulate and learn from realistic, varied workload scenarios, thereby refining the cost model's accuracy and applicability in real-world DSP applications. This strategic enumeration of parallelism, grounded in actual system characteristics and demands, ensures that the training data is not only comprehensive but also highly representative of operational conditions.

Explore around estimate parallelism...

...for meaningful data for training.

Algorithm 1 : OptiSample Strategy for Training.

```
input   : input source event rate $In_{ER}(\omega_{So})$
          Operator graph comprising set of operators $\omega \in \Omega$
output  : Parallelism degree $P(\omega)$

1  for all $\omega_i \in \Omega$ do
2      if $\omega_i$ is not $\omega_{So}$ then
3          $\omega_j = \omega_i.downstream()$ ;
4          $CurrentSelectivity(\omega_i) = EstSelectivity(\omega_i)$ Definitions 14 to 16;
5          $Out_{ER}(\omega_i) = CurrentSelectivity(\omega_i) \cdot In_{ER}(\omega_i)$ Definition 13;
6          $In_{ER}(\omega_j) = Out_{ER}(\omega_i)$;
7          if $\omega_i.upstream()$ is $\omega_{So}$ then
8              $P(\omega_i) = EstParallelism(\omega_i, In_{ER}(\omega_{So}))$ Definition 17;
9          else
10             $P(\omega_i) = EstParallelism(\omega_i, In_{ER}(\omega_i))$ Definition 18;
11     else
12         $In_{ER}(\omega_{So}) = ComputeSourceER(\omega_{So})$
```

In Algorithm 1, depicted in Lines 1–12, we detail the workings of the OptiSample strategy, which strategically assigns parallelism degrees to operators within DSP system in a methodical, bottom-up approach. This strategy begins by considering the input event rate, $In_{ER}(\omega_{So})$, of the source operator ω_{So}, and it progressively assigns parallelism degrees to all downstream operators, terminating its process at the data sink (S_I).

Parallelism degree decreases...

The source operator, ω_{So}, uniquely situated without any upstream dependencies, serves as the starting point for parallelism assignment. The parallelism degree for ω_{So} is directly derived from the application-specified event rate, and it sets the foundation for the computation of parallelism degrees for all subsequent operators. Each operator $\omega \in \Omega$ is then evaluated based

...with decreasing event rate...

on its estimated selectivity, as outlined in Lines 3-4 and referenced in Equations (14) to (16). Following the determination of selectivity, the algorithm estimates the output rate of each operator, which then becomes the input rate for the next operator downstream, as noted in Line 5 and Line 6. This cascading computation ensures that each operator's parallelism degree is informed by the output demands placed on it by its predecessor, thereby maintaining a balance between input load and processing capacity.

...for downstream operators.

Significantly, if the operator is the source operator (checked in Line 11), its parallelism degree is directly tied to the original event rate, serving as a baseline for the parallelism calculations of downstream operators, as seen in Line 12 and Line 7. This systematic enumeration of parallelism degrees for each operator ensures the determination and exploration of meaningful parallelism degrees around determined parallelism degrees across the operator graph. In the following section, the formal definitions and equations below further explain the parameters and processes of the OptiSample strategy.

Definition 13. *Output rate of an arbitrary operator* ω: The output rate, denoted as $Out_{ER}(\omega)$, is calculated based on the input rate received either from the upstream operator, $Out_{ER}(\omega_i)$, or directly from the source, $In_{ER}(\omega_{So})$, in scenarios where ω is the first operator in the sequence. This output rate calculation incorporates the selectivity of the operator to accurately determine the volume of data processed. Specifically, for a given upstream operator ω_i and its downstream counterpart ω_j, the output rates are computed as follows:

$$Out_{ER}(\omega_i) = In_{ER}(\omega_i) \cdot \text{sel}(\omega_i), \text{ where } \omega_i = (\omega_\sigma, \omega_\bowtie, \omega_\xi), \tag{13}$$

$$Out_{ER}(\omega_j) = \begin{cases} Out_{ER}(\omega_i) \cdot \text{sel}(\omega_j) & \text{, if } \omega_i = (\omega_\sigma, \omega_\bowtie, \omega_\xi) \\ In_{ER}(\omega_{So}) \cdot \text{sel}(\omega_j) & \text{, if } \omega_i = \omega_{So}. \end{cases}$$

Definition 14. *Selectivity of filter operator* ω_σ: The selectivity of a filter operator, denoted as $sel(\omega_\sigma)$, is defined as the ratio of the number of tuples that satisfy the filter condition to the total number of tuples in the input data stream. Specifically, for the filter operator ω_σ, selectivity is calculated by dividing the number of tuples $|f_{\omega_\sigma}(D_{In})|$ that pass the filter criteria by the total number of tuples $|D_{In}|$ in the input data stream D.

This relationship is represented by the following equation:

$$sel(\omega_\sigma) = \frac{|f_{\omega_\sigma}(D_{In})|}{|D_{In}|}, \quad \text{with } 0 \leq sel(\omega_\sigma) \leq 1. \tag{14}$$

Definition 15. *Selectivity of the operator* $\omega_{\bowtie}$: The selectivity of a join operator, denoted as sel($\omega_{\bowtie}$), is calculated based on the proportion of tuples that successfully join from two input streams. Specifically, for the join operator $\omega_{\bowtie}$, this selectivity is determined by the ratio of the number of tuples resulting from the join operation between two windows $|W_{D1_{In}} \bowtie W_{D2_{In}}|$, to the total number of tuples possible in the Cartesian product of the tuples in the input windows $|W_{D1_{In}}|$ and $|W_{D2_{In}}|$ from the respective input data streams D_1 and D_2.

This relationship is captured in the following formal representation:

$$sel(\omega_{\bowtie}) = \frac{|W_{D1_{In}} \bowtie W_{D2_{In}}|}{|W_{D1_{In}}| \times |W_{D2_{In}}|}, \quad \text{with } 0 \leq sel(\omega_{\bowtie}) \leq 1. \tag{15}$$

Definition 16. *Selectivity of aggregation operator* ω_ξ: The selectivity of the window aggregate operator, denoted as sel(ω_ξ), is defined by the ratio of distinct group-by tuples within a specified window of the input stream to the total number of tuples within that window. Specifically, for the window aggregate operator ω_ξ, this selectivity is calculated as the fraction of unique group-by tuples in the window $W_{D_{In}}$ relative to the total count of tuples in the same window, $|W_{D_{In}}|$.

This quantifies the reduction in data volume affected by the aggregation based on group-by attributes. This relationship can be formally expressed as follows:

$$sel(\omega_\xi) = \frac{|\textit{group-by } (W_{D_{In}})|}{|W_{D_{In}}|}, \quad \text{with } 0 \leq sel(\omega_\xi) \leq 1. \tag{16}$$

Based on the estimated selectivities, input and output rates, we define how we select parallelism degrees for training.

Definition 17. *Parallelism degree of an arbitrary upstream operator* ω_i: The parallelism $P(\omega_i)$ of the upstream operator is calculated by applying a scaling factor (sf), which is established through empirical analysis to identify conditions leading to backpressure in streaming operators. This scaling factor is used to adjust the parallelism in proportion to the input event rate $In_{ER}(\omega_i)$ received by the upstream operator.

$$P(\omega_i) = sf \cdot In_{ER}(\omega_i) \tag{17}$$

Definition 18. *Parallelism degree of an arbitrary downstream operator ω_j after ω_i*: The parallelism $P(\omega_j)$ of the downstream operator is calculated using a scaling factor, which adjusts according to the output rate $Out_{ER}(\omega_i)$ from the preceding upstream operator ω_i.

This relationship is formalized in the equation below:

$$\begin{aligned} P(\omega_j) &= sf \cdot In_{ER}(\omega_j) \\ &= sf \cdot Out_{ER}(\omega_i) \\ &= sf \cdot In_{ER}(\omega_i) \cdot \text{sel}(\omega_i) \text{ (By Equation (13))} \end{aligned} \tag{18}$$

The constraints for equations Equation (17) and (18) ensure that the parallelism for each operator, both up and downstream (ω_j, ω_i), does not exceed the predefined maximum parallelism max_P, which in turn is limited by the number of processing cores (n_{core}) available on the physical resource, i.e., $\max_P \leq \text{n}_{core}$. Additionally, it is important to note that the parallelism degrees used in training data are based on estimated selectivities and output rates rather than actual observed values. This approach is designed to balance exploration and exploitation in the training data, allowing the model to learn from both efficient and less efficient PQP configurations when the estimations deviate from actual performance.

5.6 Summary

This chapter introduces our second key contribution to this thesis, ZeroTune —a novel performance prediction model, designed to predict accurate performance for seen and unseen PQP, addressing the second research question (refer to RC2 in Section 1.2). This method circumvents the costly trial-and-error adjustments typically required at the onset of query execution. ZeroTune offers a solution to the limitations of existing methods for parallelism tuning and performance estimation, which typically rely on runtime performance monitoring and require multiple adjustments to parallelism, or existing machine learning methods that necessitate iterative retraining for each new dataset or substantial data collection to achieve generalization. ZeroTune employs a zero-shot cost model to predict accurate performance in parallel and distributed stream processing systems and integrates an *optimizer* to fine-tune initial parallelism cost-effectively. It leverages *transferable features* and a *parallel graph representation* to ensure generalizability across unseen parallel query plans, diverse workloads, and resource configurations. Additionally, ZeroTune incorporates OptiSample, data-efficient training strategy that controls the exploration of parallelism across various conditions, thereby gathering meaningful data for training while minimizing effort and enhancing generalization.

In Chapter 6, we present results from an extensive evaluation that demonstrates the robustness and accuracy of ZeroTune across various scenarios, including seen and unseen workloads, different parallelism degrees, and operator parameters (cf. Section 6.3). We also highlight its data efficiency in training and the performance improvements achieved through its parallelism tuning approach compared to other methods [83, 90, 225].

6 Evaluation of Performance Modeling Methods

The final objective of this thesis is to assess how effectively the proposed performance modeling methods perform in real-world scenarios. To achieve this, we have conducted a comprehensive evaluation of the proposed methods, PDSP-BENCH (cf. Chapter 4) and ZEROTUNE (cf. Chapter 5). We provide a detailed description of the evaluation setup in Section 6.1, including information on the specific DSP system and real-world resources that we utilize for performance benchmarking and forecasting, which are crucial for tuning parallelism in DSP systems.

Evaluation in real-world scenario...

Additionally, we outline the evaluation objectives for PDSP-BENCH and ZEROTUNE in Sections 6.2 and 6.3 respectively, specifying the goals we aim to achieve through our evaluation of each method. Through our extensive evaluation, we intend to explore the impact of various workloads (queries and data streams) and hardware configurations on the performance of DSP system under varying parallelism. We present these findings in Section 6.2, discussing the results of benchmarking the PDSP-BENCH system. Following this, we extensively evaluate the accuracy and generalizability capability of ZEROTUNE for performance predictions and the initial parallelism tuning on seen and unseen workloads across varying parallelism degrees, and query parameters. Furthermore, we demonstrate the efficiency of our proposed data-efficient training method compared to existing baselines [225, 83]. We also highlight the efficiency of ZEROTUNE relative to non-transferable model architectures [90, 120, 59], and the significant speed-ups achieved with our parallelism tuning approach compared to other methods [225, 83, 140] in Section 6.3.

...with different DSP workloads and resources.

6.1 Evaluation Setup

This section presents the evaluation environment to assess the performance modeling methods PDSP-BENCH and ZEROTUNE. In addition, we provide details on DSP system used for workload generation and research testbed used for resource provisioning and deployment of DSP system. We further discuss

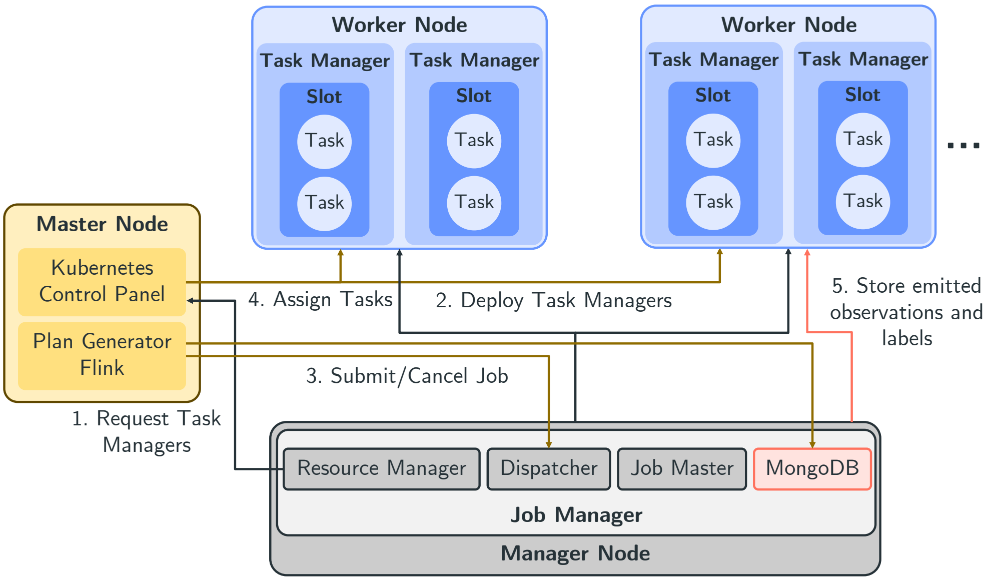


Figure 33: Apache Flink architecture showing the distribution of services and communication between job manager and task manager in a distributed environment.

details on the evaluation parameters and performance metrics used for assessing the capabilities of PDSP-Bench and ZeroTune.

6.1.1 *Distributed Environment Setup*

In the following we first discuss the workload generation in our setup and the cloud resources used for the evaluation.

Workload Generation and Resource Management

Flink for PQP and workload generation

For our research, we employ *Apache Flink* version 1.16 [52], a widely-used open-source distributed stream processing (DSP) system, to implement *Plan generator* which generates various workloads and parallel query structures (PQP) using different parallel enumeration strategies. *Apache Flink* is developed in Java, a choice supported by its extensive adoption to research and industry, robust community support, and comprehensive documentation [82]. To tailor Apache Flink for our specific research needs, we also modify its source code to collect performance measures necessary for benchmarking, training, and testing performance models. Although, our choice of DSP is Apache Flink but the proposed performance modeling methods are not limited to *Apache Flink*; can be exchanged by any DSP such as Apache

Cluster Type	Clusters	Nodes	CPU	RAM (GB)	Disk (GB)	Intel Processor	Speed (Ghz)	S/U
Ho	m510	270	8	64	256	Xeon D	2.0	S
Ho	c6420	72	32	384	1024	Skylake	2.6	U
He	rs620	38	8-10	128-384	900	Xeon	2.2	S
He	c8220x	4	20	256	4096	Ivy Bridge	2.2	U
He	c8220	96	20	256	2048	Ivy Bridge	2.2	U
He	dss7500	2	12	128	120	Haswell	2.4	U
He	c6320	84	28	256	1024	Haswell	2.0	U
He	c6525_25g	144	16	128	480	AMD EPYC	2.2	U
He	rs6525	32	64	256	1600	AMD EPYC	2.8	U

Table 6: Cluster of resources (metal servers) utilized for evaluation on CloudLab tested [76, 8, 7]. These resources are combined for performance benchmarking and performance prediction process(data generation, model training (S), and inference (S and U)).“Ho” and “He” denote homogeneous and heterogeneous resources, “S” and “U” indicate seen, and unseen dataset.

Storm [124]. Subsequently, we modify Apache Flink by injecting code to extract crucial data directly from its internal mechanisms. This data is then processed by the *Stream Monitor* to simulate data streams from different applications, then organized and stored in a central *MongoDB* database situated on a non-worker node. From there, the *Plan Generator* within *Graph Builder* of Apache Flink can access the data for further processing.

Plan generator for workload generation

Apache Flink’s architecture comprises three main components distributed across various physical nodes to enhance scalability and efficiency as presented in Figure 33. The first component, the *Plan Generator Flink*, serves as the client but is not part of the runtime environment of Apache Flink. However, it plays a critical role in generating queries, transforming stream graphs into job graphs, and forwarding them to the job manager. The *job manager*, a second component, orchestrates the execution of these graphs by converting them into execution graphs and managing resource allocation. It controls task execution and handles scaling operations through interaction with *Kubernetes*, a container orchestration system, managing Task Managers or terminating unnecessary instances. The third core component, the *Task Managers*, is responsible for actual data processing. These managers handle multiple tasks, manage data caching, and direct the processed data to appropriate destinations. Each node can operate several Task Managers, and each manager may run multiple tasks within its allocated slots, sharing the available resources efficiently.

For resource management, PDSP-Bench and ZeroTune utilize *Kubernetes* to deploy a customized Apache Flink setup as a *Docker* [182] image. This integration allows the *Job manager* to seamlessly initiate or terminate *Task Managers* as needed. Furthermore, we leverage Apache Flink’s fairness policy and job distribution capabilities to ensure efficient resource allocation and optimized performance during our evaluations. Combining these tools and strategies allows us to thoroughly assess the efficiency and effectiveness of the proposed performance modeling methods.

Kubernetes for managing cloud resources

While Apache Flink can run on Kubernetes, Apache Hadoop YARN and in standalone clusters, we choose *Kubernetes* [55] due to its enhanced scalability, and growing adoption within the Flink community. Although Apache Flink has historically been deployed more frequently on *Yarn* [240], primarily due to mature ecosystem of *YARN*, *Kubernetes* offers several advantages. It provides better scalability, more efficient resource allocation, and node management strategy, making it well-suited to meet the scalability and efficiency demands of our research. These factors make *Kubernetes* the optimal choice for resource management for our distributed stream processing workloads.

CloudLab for Resource Provisioning

CloudLab to setup distributed environment

All experiments are conducted using the *Geni* [98] and *CloudLab* research testbed [76], which provide the necessary real-world scenario of distributed cloud infrastructure with both homogeneous and heterogeneous bare metal hardware to configure and deploy an Apache Flink [52] cluster. We select CloudLab testbed as it offers an *open-source* scientific infrastructure for researchers. Also, it provides hard isolation from other users and evaluations so that hundreds of experiments can be conducted in isolation without getting affected by other experiments. In addition, it offers the possibility to deploy different software stacks such as Apache Flink, Hadoop, Yarn, and Kubernetes to build our own distributed environment from the ground. The setup enables us to perform experiments and evaluate the performance of ZERO-TUNE for performance prediction as well as PDSP-BENCH for benchmarking capabilities in real-world scenarios.

6.1.2 PDSP-BENCH *Evaluation Parameters*

Benchmarking on a wide spectrum of resources

We conduct evaluations for performance benchmarking using the PDSP-BENCH on the CloudLab research testbed to utilize distributed infrastructure, equipped with homogeneous and heterogeneous hardware configurations. This setup facilitates the deployment of a System Under Test (SUT) on CloudLab cluster as shown in Table 6, enabling comprehensive evaluations across various workload and resource configurations as shown in Table 7. For the evaluations, we opt for Apache Flink v1.16.1 [52] as SUT; however, the system allows for substitution with any DSP systems. In addition, *Apache Kafka* is deployed on a separate CloudLab cluster node to manage

Diversity	Parameters	Parameter Data Range
Query	Real-world query structures	Word Count (WC), Machine Outlier (MO), Linear Road (LR), Logs Processing (LP), Google Cloud Monitoring (GCM), TPC-H, Bargain Index (BI), Sentiment Analysis (SA), Smart Grid (SG), Click Analytics (CA), Spike Detection (SD), Trending Topic (TT), Traffic Monitoring (TM), Ad Analytics (AD) (cf. Table 4)
	Synthetic query structures	Linear, 2-chained filter, 3-chained filter, 4-chained filter, 2-way join, 3-way join, 4-way join, 5-way join, 6-way join
	Parallelism degree categories	$1 \leq XS < 8$, $8 \leq S < 16$, $16 \leq M < 32$, $32 \leq L < 64$, $64 \leq XL < 128$, $128 \leq XXL$
	Window duration (ms)	50, 100, 150, 200, 250, 325, 750, 1k, 1.5k, 2k, 2.5k, 3k, 4k, 5k, 6k, 7k, 8k, 9k, 10k
	Window length (tuples)	2, 3, 4, 5, 7, 10, 17, 25, 37, 50, 62, 75, 82, 100, 150, 200, 250, 300, 350, 400
	Sliding length (ratio)	[0.3, 0.4, 0.5, 0.6, 0.7] x Window length
	Window types and policy	type: sliding and tumbling, policy: count and time-based
	Window aggr. functions	min, max, avg, mean, sum
	Join and filter data types	string, integer, double
	Filter functions	$\leq, \geq, \neq, =, <, >$, startsWith, endsWith, endsNotWith, startsNotWith
Data	Tuple width and data type	[1 - 15] x [str., doubles, int]
	Event rate (events/sec)	10, 100, 1k, 5k, 10k, 50k, 100k, 200k, 500k, 1mn
	Partitioning strategy	Strategy for data distribution (forward, rebalance, hashing)
Resource	Cluster type	Homogeneous: m510, Heterogeneous: c6320, c6525_25g

Table 7: PDSP-Bench benchmark parameters for SUT (order by complexity) [7].

data flow, producing data at varying event rates through an *input topic* and consuming the output from the SUT using an *output topic*. PDSP-Bench utilizes various range of evaluation parameters as mentioned in Table 7 for benchmarking the performance of SUT for various data stream, PQP and resources configurations.

Data Generation for Benchmarking

PDSP-Bench generates different PQP for various query structures ($\approx 30k$ PQPs) including both synthetic ($\approx 20k$) and real-world ($\approx 10k$) applications. These PQP from diverse query structures are executed in three cycles, each lasting three minutes on clusters comprising ten nodes. Performance metrics are collected and stored locally in a *MongoDB* database to facilitate detailed analysis. The highest event rate for evaluation is *four* million events per second; a scale is chosen based on the presumption that higher event rates significantly benefit from increased parallelism, although results for various rates are available. Subsequently, the generated workload configuration and corresponding per-

Benchmark using different synthetic and real-world applications formance as labels are used by ZeroTune models for training and testing the accuracy of the cost models.

PDSP-Bench *Controller and Web User Interface (WUI)*

WUI to ease benchmarking process

A *controller* of PDSP-Bench is implemented in *Django*, along with a Web User Interface (WUI) developed using *Vue.js*, to manage user inputs regarding cluster setup, SUT deployment, and PQP execution, as outlined in system overview of PDSP-Bench (cf. Section 4.3.1). This interface ensures that users can easily configure and manage the benchmarking process, enhancing the usability and accessibility of PDSP-Bench.

6.1.3 ZeroTune *Evaluation Parameters*

Seen data for training

To gather the training and testing benchmarks, we implement PQP and workload generator on top of Apache Flink, focusing on various parameters summarized in Table 8. Our primary focus is on the training range using the observed cluster nodes detailed in Table 6.

Unseen data for testing

Seen and unseen data are disjunct

As mentioned earlier, all experiments are conducted using the *CloudLab* research testbed [76] comprising infrastructure that allows configuring and deploying an Apache Flink [52] cluster. The setup enables us to perform experiments and evaluate the performance of ZeroTune effectively. Table 8 outlines the selected training and testing ranges to evaluate the ZeroTune model, based on a dataset comprising $24k$ synthetic queries. These queries are categorized into three types: linear, 2-way joins, and 3-way joins, with each category containing $8k$ queries. The dataset is divided into training (80%), testing (10%), and validation (10%) sets.

For the training strategy of ZeroTune, we employ OptiSample and compared its performance with random sampling wherever explicitly specified (cf. Section 5.5). Cluster management and task placement are handled using Apache Flink's task manager, with Kubernetes utilized for orchestration.

6.1.4 *Performance Metrics*

For performance benchmarking using PDSP-Bench, we focus on *end-to-end latency* (cf. Definition 6) for query execution of SUT, i.e., Apache Flink. We benchmark SUT for other performance metrics as well, such as throughput, resource utilization, etc. Due to the high amount of experiment data,

Parameters	Unit	Training Range (Seen)	Testing Range (Unseen)
Event rate	ev/sec.	100, 200, 400, 500, 700, 1k, 2k, 3k, 5k, 10k, 20k, 50k, 100k, 250k, 500k, 1m	50, 75, 150, 300, 450, 600, 850, 1.5k, 4k, 7.5k, 15k, 35k, 175k, 375k, 750k, 1.5m, 2m, 3m, 4m
Tuple width and data type	values	[1 - 5] x [str., doubles, int]	[6 - 15] x [str., double, int]
Window length	tuples	5, 10, 25, 50, 75, 100	2, 3, 4, 7, 17, 37, 62, 82, 150, 200, 250, 300, 350, 400
Window duration	ms	250, 500, 1k, 2k, 3k	50, 100, 150, 200, 325, 750, 1.5k, 2.5k, 4k, 5k, 6k, 7k, 8k, 9k, 10k
Sliding length	ratio	[0.3, 0.4, 0.5, 0.6, 0.7] x Window length	
Cluster type	-	ms510, rs620	c6420, c8220x, c8220, dss7500, c6320, rs625
Network link speed	Gbps	1, 10	
Number of workers	-	2, 4, 6	3, 8, 10
Parallel query structures	-	Linear, 2-way join, 3-way join	2-chained filters, 3-chained filters, 4-way join, 5-way join, 6-way join
Benchmark queries	-	-	Spike detection, Smart-grid (local), Smart-grid (global)
Operator type	-	Source, Filter, Window-join, Window-Aggregation	
Parallelism degree categories	-	$1 \leq XS < 8$, $8 \leq S < 16$, $16 \leq M < 32$, $32 \leq L < 64$, $64 \leq XL < 128$	

Table 8: Training and testing ranges [8, 5, 6] for data generation and inference with ZeroTune. The seen range evaluates overall model performance using a traditional training-test split, while the unseen range tests the generalizability of the model. Extrapolation parameters are underlined, with the remaining unseen parameters are used for interpolation.

we are presenting some of the selected metrics and corresponding observations. In Section 6.2, we present benchmarking results for end-to-end latency and in Section A.1.4, we show resource usage (cf. Definition 10) benchmark results for SUT. For performance benchmarking of the accuracy of learned cost models, we use Q-error as described below for ZeroTune. We report the mean of three runs of measuring median end-to-end latency (*50th percentile*) in our evaluation for performance benchmarking.

Q-error to measure prediction accuracy

Furthermore, to measure the accuracy and generalization capabilities of ZeroTune, we employ the Q-error metric $q(c, c')$, where $q(c, c') \geq 1$. The metric, which is widely recognized in the field [149], quantifies the relative deviation between the true cost c (in terms of latency and throughput) and the predicted cost c'. The accuracy of the ZeroTune model is evaluated using this Q-error metric for both latency and throughput, following the OptiSample training procedure. Our experimental results indicate that the q-error values are predominantly close to 1, suggesting that the ZeroTune model

Q-error close to one means high accuracy provides accurate cost predictions and demonstrates strong generalizability across different scenarios.

Finally, for evaluating the *optimizer*, we employ a weighted cost metric (cf. Equation (12)) and calculate the *mean speed-up*. The mean speed-up is determined as the fraction of improvement in latency or throughput compared to baseline measurements. These metrics collectively highlight the effectiveness and efficiency of the ZeroTune model in optimizing performance.

6.1.5 *Baselines for Performance Comparison*

For PDSP-Bench, no baseline for comparison is required, but the capabilities of the performance benchmarking are shown on DSP system, Apache Flink.

Comparison with baselines for...

To show the prediction accuracy and generalization capability of ZeroTune, first, we compare ZeroTune to other *non-transferable architectures*, flat vector representations from DBMS [114]. With this evaluation, we intend to show the impact of model architectures on learning patterns. This baseline approach represents features like the number of different operator types, their selectivity, and the degree of parallelism, which are included as a flat vector. We train a linear model based on flat vector representation, drawing inspiration from [90]. We implement the linear model using Linear Regression [20] and extend it to a *deep neural network*, creating a Flat Vector MLP (Multilayer Perceptron) [21] that learns from the flat vector representation. Additionally, we train a *random forest model* [45] using the same flat vector approach.

...prediction accuracy and generalization.

Furthermore, we compare ZeroTune against a *greedy approach* [225] and *Dhalion* [83], an optimizer that incorporates analytical derivation of parallelism degrees. The comparisons are made in terms of *mean speed-ups* and *weighted sum metrics* (cf. Equation (12)). These comparisons assist us in assessing the relative performance and effectiveness of ZeroTune in determining initial parallelism and generalizing cost predictions for DSP systems.

6.2 Experiments on PDSP-Bench

Understanding effect of parallelism...

In this section, we present our evaluation for performance benchmarking utilizing PDSP-Bench. The evaluation of system involves extensive experiments that highlight its capabilities in benchmarking parallel stream processing. By using Apache Flink as the SUT, the evaluation demonstrates the impact of varying parallel query complexities, hardware configurations, and workload parameters on system performance. The results show the importance of considering both parallelism and heterogeneity to achieve optimal performance in real-time data processing applications. Furthermore, we integrate and train various learned cost models for streaming queries, demonstrating their performance in terms of both model accuracy and training efficiency. Considering the extensive range of workloads, including 14 synthetic and 9 real-world applications, along with the various parameters within PDSP-Bench as shown in Table 7, we have selected specific observations (**O#**) from our evaluations.

Exp. 1: Impact of PQP complexity on performance. We examine how increasing the complexity of parallel query structures (PQP) influences system performance. Specifically, we examine the effects of different types of operators as well as the degree of parallelism. By analyzing these factors, we aim to understand how complex query structures impact overall performance metrics.

Exp. 2: Impact of heterogeneous hardware on performance. We explore how the use of heterogeneous hardware resources affects the execution of PQP. By comparing the performance of PQP across different hardware configurations, we aim to identify the benefits and challenges associated with deploying queries in heterogeneous systems. This analysis helps to understand how variations in hardware resources can influence query processing efficiency and execution times.

...hardware diversity...

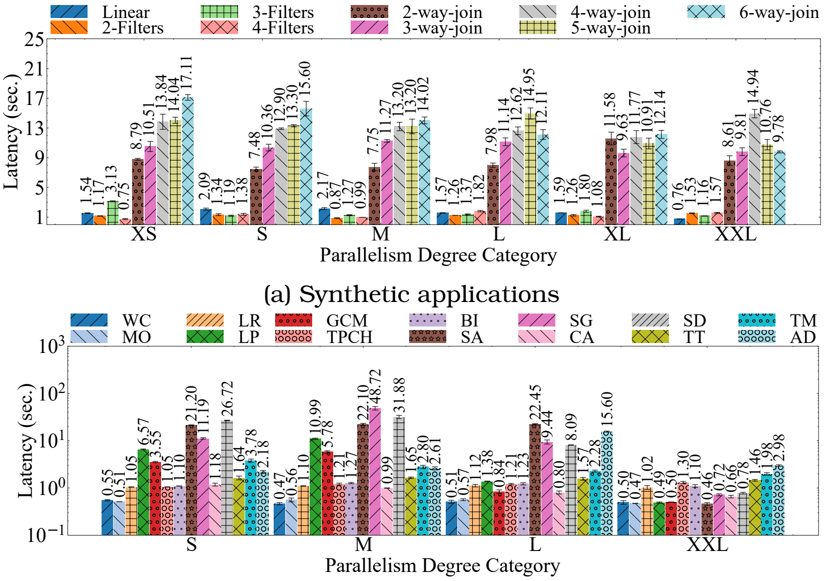
Exp. 3: Integration of ML models in PDSP-Bench. We evaluate how different learned cost models perform on various PQP and how their prediction accuracy varies depending on PQP complexity.

...on performance of PQP.

6.2.1 *Workload Diversity*

PQP with varying complexity

Exp. 1: Impact of PQP complexity. We benchmark Flink using the presented categories of PQP: a) synthetic PQP mainly comprising standard SPS operators (cf. Figure 34 *top*) and b) real-world PQP comprising both standard SPS and user-defined operators (UDOs) (cf. Figure 34 *bottom*). We select homogeneous resources of `m510` cluster (with 10 nodes) to analyze *only* parallelism degree diversity. Here, the complexity of a PQP correlates both the composition of various operators and the parallelism degree applied to execute them. For instance, *linear* PQP, which includes a single source, multiple filters, and a window aggregation without joins, represents a simpler data flow. This simplicity typically leads to lower computational demands and, consequently, lower end-to-end latency. However, the presence of dual filters introduces computational requirements that can affect latency based



(a) Synthetic applications

(b) Real-world applications

Figure 34: Impact of parallelism degree on performance of PQP from *synthetic* (top) and *real-world* applications (bottom) [7]. The comparative analysis reveals distinct performance behaviors, illustrating both the beneficial aspects of parallelism in improving end-to-end latency and the need for careful consideration of queries and operators characteristics within PDSP environments. The results underscore a complex interplay between standard operations and user-defined operators (UDOs), marked by both *paradoxical* effects and *non-linear* performance trends in relation to the parallelism degree. (Note: The bottom figure omits XS and XL; their performance mirrors S and L).

on the data volume and filter complexity. On the other hand, the complexity significantly increases with multi-way joins, starting from 2-way joins. Each join operation, requiring the correlation of data from multiple sources, adds substantial computational and data management overhead, which requires parallel processing while managing the increased latency implications effectively. We present interesting observations of this analysis as follows.

Parallelism assists in improving performance

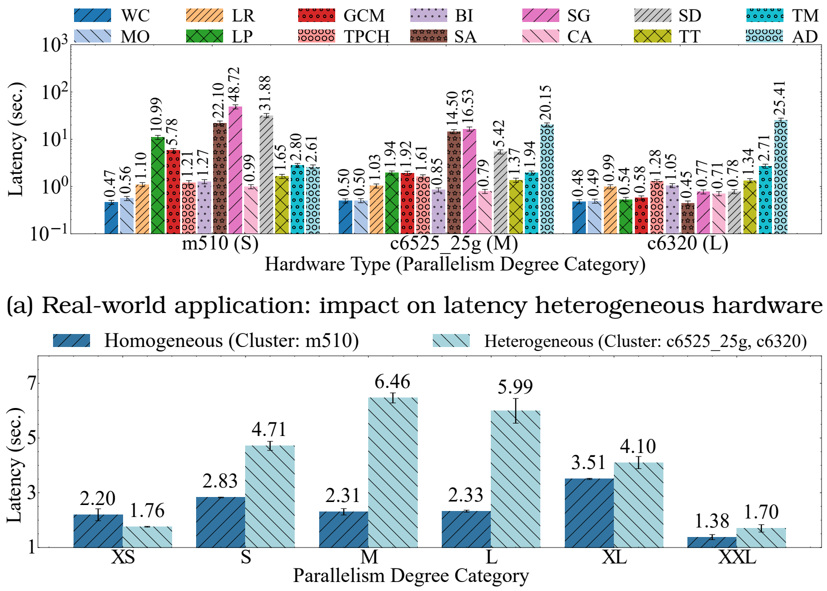
O1 *- Increasing parallelism can speed-up multi-way join queries.* We observe an interesting trend in Figure 34 (top) when PQP transitioned from linear query to more chained filters and joins. Initially, when we add more filters, the latency remains almost consistent with increasing parallelism categories, i.e., *XS to XXL*, reflecting the minimum parallelism is sufficient to handle query complexity. However, when we introduce join operators in PQP, it represents a tipping point where latency increases linearly. This reflects the inherent complexity of coordinating and executing join operations across distributed datasets. At the same time, the parallel instances assist in handling workload and reducing latency with increasing complexity. A similar trend is also observed for PQP from real-world applications in Figure 34 (bottom) where PQP consist of combinations of varying numbers and types of standard SP operators and UDOs. For instance, applications with more standard DSP operators such as `WC`, `LR` showed a consistent performance while PQP with more computation intensive UDOs such as *SA, SG, SD* has shown significant performance improvement with increasing parallelism degrees. Thus, it highlights that parallelism is particularly beneficial for PQP with data-intensive operators, significantly enhancing performance compared to PQP with complex structures but less data-intensive operators.

O2*- Increasing parallelism can speed-up queries, but not in all cases.* While increasing parallelism resulted in enhancing the performance by distributing the workload, for some PQP, there is a paradoxical effect on end-to-end latency as PQP complexity increases. Beyond a certain threshold of parallelism, i.e., $32 \leq L < 64$, particularly with multiple joins, the overhead of managing parallel operations—such as data shuffling and synchronization can outweigh the benefits, leading to increased latency. This scenario is comparable to traffic flow in a city where, beyond a certain volume, adding more lanes (parallel paths) can actually lead to more congestion due to bottlenecks at merge points and intersections. Thus, performance improvements in multi-way joins as parallelism increases from L to XL are small or negligible. In contrast, the PQP from real-world applications starts showing the benefit of parallelism on performance improvement in latency when the parallelism degree becomes extremely high. For instance, in the parallelism category $32 \leq L < 64$, PQP from `SG` and `SD` show significant performance improvements. In contrast, `AD` shows negligible performance gains even with parallelism greater than 128, highlighting the complexity of the operators in the query.

Not all PQP get benefit of increasing parallelism

O3*- Queries with UDOs shows unpredictable performance.* In our evaluation, we also investigate the performance intricacies of standard SP operators compared to UDOs within PDSP environment. Our findings reveal distinct scalability and computational overhead that underline the relationship between operator types and parallel processing efficiency. Standard SP operators exhibit predictable scalability due to their well-defined semantics. For instance, a *flatMap* operation in a `WC` application scales almost linearly with the increase in parallelism, as it can be independently executed across different partitions of the data without the need for complex state management or coordination. On the other hand, UDOs, which allows for embedding of custom logic into the data processing pipeline, introduces variability in performance scalability, primarily due to the potential for complex state handling and the need for coordination across different execution units. For instance, in the `AD` application, the presence of multiple UDOs performing custom aggregation and joining logic on a sliding window introduces a non-linear scaling behavior. As parallelism increases, the overhead of managing state consistency and communication between parallel instances leads to a sublinear performance

Operators processing logic can affect processing time increase, and in some cases, performance may even degrade due to the overhead exceeding the computational benefits of parallelism.



(a) Real-world application: impact on latency heterogeneous hardware

(b) Synthetic application: impact on latency heterogeneous hardware

Figure 35: Impact of heterogeneous hardware on performance for PDSP, with varying parallelism degree and resource processing capabilities on real-world (top) and synthetic (bottom) applications [7]. The evaluation demonstrates that parallel processing benefits from hardware diversity in enhancing performance. However, achieving optimal performance requires a thorough understanding of hardware characteristics and effective workload distribution to avoid pitfalls such as the diversity dilemma.

***O4**- The non-linear effect of parallelism on latency:* PDSP-Bench provides critical insight into the non-linear relationship between the parallelism degree of PQP and performance. As the parallelism of standard SP operators in a query increases, the performance does not always linearly improve since not every PQP benefits equally from parallelism. This is evident from observations O1 - O3, where performance is influenced by factors such as operator complexity and the paradox of parallelism. For instance, queries like *SA*, *SG*, *SD* exhibit high latency in parallelism categories S and M, but start showing improvements at level L, with significant performance gains when parallelism exceeds 128.

Non-linear effect of parallelism on performance

6.2.2 *Hardware Diversity*

Exp. 2: Impact of heterogeneous hardware on performance. Next, we evaluate the impact of homogeneous (`m510`) and heterogeneous hardware (`c6525_25g,c6320`) clusters on performance for PDSP and execute PQP across clusters of 10 nodes each. Figure 35 (top) represents the mean *end-to-end latency* of PQP with parallelism degree category as per number (#) of cores on hardware of each cluster to analyze the performance of real-world applications. For instance, `m510` cluster has hardware with 8 cores, so selected PQP with S parallelism degree category. Similarly, PQP with parallelism degree categories M and L as clusters `c6525_25g` and `c6320` have hardwares with 16 and 28 cores, respectively. Figure 35 (bottom) shows the mean *end-to-end latency* across different parallelism categories of PQP for three types of clusters, specifically for synthetic applications.

Parallelism relation with resources heterogeneity

***O5**- Powerful heterogeneous environment does not necessarily accelerate queries.* By evaluating PDSP across diverse hardware configurations, we encounter the *diversity dilemma* where the theoretical advantages of heterogeneous environments sometimes clash with practical performance outcomes as shown in Figure 35 (top). We anticipate that the diversity in computational processing capabilities would universally accelerate the parallel processing and enhance the performance. However, we notice that while applications like *SA*, *CA*, and *SD* significantly benefit from the increasing processing power of underlying hardware and can achieve an exponential decrease in latency. On the other hand, *AD* struggles to improve the performance in heterogeneous hardware configuration due to the complexity of UDOs coupled with the communication overhead across different instances. This finding suggests that the theoretical benefits of hardware diversity require careful orchestration of workload distribution and resource management strategies to leverage heterogeneous environments effectively.

Processing powers help in accelerating performance...

...but not in every scenario.

***O6**- Finding optimal parallelism for queries is non-trivial.* We also evaluate parallel processing capabilities of synthetic PQP on various hardware clusters as presented in Figure 35 (bottom). We notice that there is no consistent balancing point of parallelism for all workloads. Until this point, parallelism might or might not yield further performance benefits or might even hinder it due to increased communication and synchronization overhead between parallel instances. It might be even more challenging to find this point in a heterogeneous environment. For instance, as parallelism increases, we observe varying performance across both homogeneous and heterogeneous clusters. In the homogeneous cluster environment, we observe a trend where latency generally increases with the parallelism category from XS to XL, after which the latency decreases for XXL. We observe a similar behavior in heterogeneous clusters. The latency is highest until the parallelism category M and decreases for larger parallelism categories (L, XL, XXL). This indicates that the heterogeneous cluster, which likely consists of nodes with different computational capabilities, benefits more significantly from increased parallelism. The initial higher latency at medium parallelism could be attributed to the imbalance in workload distribution across the heterogeneous resources. As the

Hard to find optimal parallelism...

...in heterogeneous environment.

parallelism degree grows, the system may better utilize the diverse computational resources, thus reducing the overall latency.

Diversity dilemma...

...adverse effect on performance.

***O7**- Homogeneous or heterogeneous clusters: there is no clear choice for all cases.* PDSP-BENCH provides insight about notable performance enhancement with increasing parallelism and diverse hardware configurations as shown in Figure 35 (top). Specifically, PQP from real-world applications benefit significantly from increasing parallelism and hardware processing capability, leading to improved performance. Conversely, PQP from synthetic applications demonstrate better performance on homogeneous clusters than heterogeneous ones. This pattern can be attributed to the nature of synthetic PQP, encompassing standard SP operators, which are more efficiently handled in homogeneous clusters. Conversely, heterogeneous clusters, despite their potential for high computational power, present challenges in achieving even workload distribution, increased communication and synchronization overheads due to varying communication speeds across different hardware units.

6.2.3 *Integration of ML Models*

Influence of PQP complexity...

The `ML Manager` (cf. Section 4.3) of PDSP-BENCH offers benchmarking of performance of various ML models tailored for parallel and distributed stream processing environments. It uses data collected during benchmarking as labeled datasets for training and inference. In this evaluation, we focus on assessing the effectiveness of different learned cost models in predicting the performance of streaming queries, such as end-to-end latency.

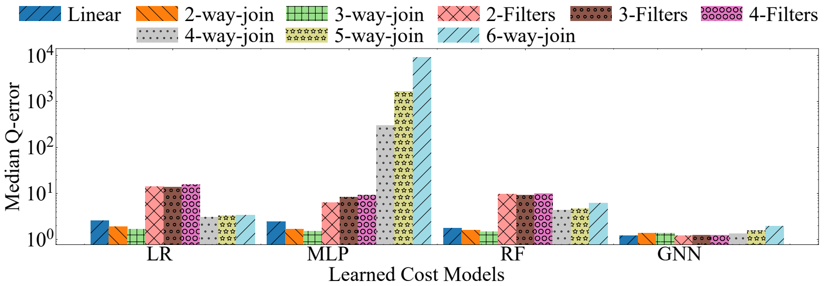


Figure 36: Comparison of performance prediction accuracy of various ML models: Linear regression (LR) [90], Multi-layer perceptron (MLP) [120], Random forest (RF) [59], Graph neural networks (GNN) [258, 8, 106] for various PQP.

...on cost models.

We integrate four distinct machine learning (ML) models architectures into the `ML Manager`:(1) Linear regression (LR) [90]: traditionally used for its simplicity and effectiveness in prediction tasks, (2) Multi-layer Perceptrons (MLP) [120]: known for capturing nonlinear relationships in data, (3) Random forest (RF) [59]: utilizes decision trees to improve prediction accuracy, and

(4) Graph neural networks (GNN) [258, 8, 106]: applies graph structures to model complex relationships in data. It encodes PQP as a DAG [8] within GNN, allowing the model to treat different operators within PQP as nodes, and the relationships between them as edges. These models are selected based on their diverse approaches to model and handle the complexities of stream processing queries.

The accuracy of these models is measured using Q-error $q(c, c')$ *where* $q \geq 1$ metric (cf. Section 6.1.4). Here, q-error being close to 1 represents better prediction accuracy. We also implement early stopping for each ML model to prevent overfitting and ensure efficient training times. We implement early stopping with a patience value of 100 epochs, meaning the training process monitors the validation loss and halts if there is no improvement for 100 consecutive epochs. This method is uniformly applied across all models to maintain consistency. The training of these models is performed on $m510$ clusters.

GNN efficiently learn and capture PQP relations

O8- *Graph representation assists in learning dynamic behavior of SPS.* The primary aim of benchmarking ML models is to evaluate how these diverse cost models perform across various streaming queries. Figure 36 indicates that the GNN model consistently surpasses other models in predicting cost, i.e., latency, even as query complexity increases. The high accuracy of GNN can be attributed to its *graph-based representation*, which effectively captures and leverages the intricate dependencies within query structures and the dynamic behaviors of stream processing systems. The graph-based represents results in more accurate and reliable performance predictions compared to other models. A similar observation is made for PQP related to real-world applications, and thus we only present results on synthetic PQP.

6.3 Experiments on ZeroTune

Extensive evaluation to understand...

In this section, we present the evaluation results of ZeroTune, focusing on its ability to efficiently generalize across unseen workloads and resource configurations, as well as its effectiveness in using cost estimates to optimize parallelism degrees. The evaluation aims to validate the robustness, accuracy, and efficiency of ZeroTune under various workloads and resource configuration conditions. Here are the specific evaluation questions that we address for ZeroTune:

- **Exp. 1: Accuracy on Seen and Unseen Workloads.** We assess the accuracy of ZeroTune for both seen and unseen parallel query structures (PQP) and benchmarks. It seeks to ascertain the prediction accuracy of the model and its generalization across varying conditions.

- **Exp. 2: Generalization Performance on Fine-Grained Parallelism.** Following up on the previous evaluation, we further assess the performance of ZeroTune across different ranges of parallelism degrees to examine how the model maintains high accuracy as the degree of parallelism varies, highlighting its adaptability to diverse and increasing parallelism.

...model accuracy on seen and unseen data...

- **Exp. 3: Generalization across Unseen Parameters.** We explore the capability of ZeroTune to predict costs for parameters not encountered during training, such as different tuple widths. We intend to test how does the model apply learned insights to predict cost for completely new and unseen workload characteristics.

...generalization capability...

- **Exp. 4: Data-Efficient Training.** We assess the training efficiency, i.e., number of queries and training time, taken by OptiSample. This experiment is crucial for determining the practicality of deploying ZeroTune in real-world scenarios where data and computational resources may be limited.
- **Exp. 5: Optimizer for Parallelism.** We determine if ZeroTune, in conjunction with its integrated *optimizer*, can effectively identify the optimal set of initial parallelism degrees that minimize the overall cost of query execution. It measures the efficacy of the optimizer in determining the parallelism degree based on predicted cost.

...and efficiency in determining parallelism.

- **Exp. 6: Feature Ablation Study.** The final experiment investigates how selected transferable features influence the generalization capabilities of ZeroTune. By selectively disabling certain features, the study aims to highlight which attributes are most critical for the ability of the model to generalize.

These experiments collectively aim to demonstrate the comprehensive capabilities of ZeroTune, showcasing its potential to accurately predict and generalize performance for diverse workloads and heterogeneous resource configurations in DSP systems.

6.3.1 *Accuracy on Seen-Unseen Workload*

Overall model comprises linear, 2- and 3-way join queries

Exp. 1: Impact of seen-unseen workload. In our initial experiment, we assess the prediction accuracy of the ZeroTune model on known or seen query structures, as outlined in Table 8, before examining its ability to generalize to novel query structures and benchmarks. Our evaluation focuses on the accuracy of latency and throughput prediction, utilizing both median and

Trained Model	**Query**	**Q-error (Latency)**		**Q-error (Tpt)**	
ZeroTune-OptiSample	**Structure**	**Median**	**95th**	**Median**	**95th**
① Seen workload	Linear	**1.21**	2.51	**1.24**	2.31
	2-way-join	**1.37**	3.84	**1.82**	8.05
	3-way-join	**1.38**	3.35	**1.89**	7.20
	Overall	**1.30**	3.35	**1.57**	6.82
② Unseen workload	2-filter-chained	**1.22**	2.15	**1.40**	3.50
	3-filter-chained	**1.24**	2.55	**1.57**	3.96
	4-filter-chained	**1.24**	2.90	**1.64**	5.31
	4-way-join	**1.34**	2.92	**1.92**	6.55
	5-way-join	**1.56**	3.6	**2.93**	16.82
	6-way-join	**1.95**	6.8	**6.19**	36.29
③ Unseen benchmark	Spike Detection	**1.29**	2.40	**2.73**	5.99
	Smart-grid (local)	**1.33**	1.50	**2.30**	4.44
	Smart-grid (global)	**1.44**	1.60	**1.52**	2.40

Table 9: Median and 95th percentile q-errors for cost prediction across seen and unseen parallel query structures using synthetic data and public benchmarks. ZeroTune model demonstrates efficient data utilization, providing highly accurate cost predictions for both known and novel workloads [8, 6].

95th percentile q-errors as indicators of performance, where a q-error of 1.0 signifies an exact prediction.

During this phase, ZeroTune model is trained and evaluated using the OptiSample training strategy, which is specifically designed to adjust parallelism degrees. This approach is contrasted with different model architectures employing flat vector representations as depicted in Figure 38.

These comparisons aim to identify which method more effectively enhances model performance across both seen and unseen workloads. The series of evaluations helps to evaluate the robustness of the ZeroTune model in adapting to diverse operational scenarios.

Seen Workload. In Table 9: ①, we analyze the PQP performance on query structures, specifically linear, 2-way joins, and 3-way joins, within the seen test set range, as outlined in Table 8. Additionally, we present the "overall" accuracy across all query structures. The results demonstrate that ZeroTune consistently delivers highly accurate cost predictions for both latency and throughput, highlighting the effectiveness of the OptiSample training strategy. The performance of ZeroTune surpasses all baseline model architectures that use flat vector representations, which fail to capture the structural flow of the query graph that ZeroTune leverages (cf. Figure 38).

High accuracy for seen PQP

For the remaining evaluations, unless specified otherwise, we focus on the *overall* model trained on all query types. The approach provides a comprehensive understanding of ZeroTune's capabilities across diverse query structures.

Unseen Workloads. A key goal of the proposed approach is to enable the ZeroTune model to generalize to completely unseen parallel query structures that are not included during the training phase. To evaluate the capability, we evaluate the model on more complex parallel query structures than those used during training. The increased complexity includes more intricate filter predicates and join operations, such as 2-4 chained filters and 4-6 way joins, as detailed in Table 8. We generate 200 test queries per query type to conduct the assessment.

ZeroTune shows high accuracy for unseen PQP

The results in Table 9: ② indicate that ZeroTune maintains high accuracy and adaptability in predicting costs, even for unseen parallel query structures. This is evidenced by both the median and 95th percentile q-errors, demonstrating the model's effectiveness in adapting to *unseen* streaming workloads right out of the box. While ZeroTune performs exceptionally well with simpler structures, such as filter chains, we observe an increasing trend in q-errors as the complexity of the unseen parallel query structures grows. This trend is particularly notable for throughput predictions, especially in scenarios involving *6-way* joins and high degrees of parallelism, where throughput increases dramatically. This highlights the challenges of maintaining prediction accuracy as query complexity and parallelism scale up.

Unseen Queries from Public Benchmarks. In addition to evaluating synthetic queries, we conduct further assessments to determine the accuracy and generalization capabilities of the ZeroTune model using two public streaming benchmark queries: *spike detection* and *smart-grid* [42, 181, 36].

High accuracy for public benchmark data

The *spike detection* benchmark [36] processes a stream of sensor data to detect spikes by comparing current values to the average values over the previous 2 seconds. For the evaluation, we generate artificial sensor data to simulate the benchmark conditions. The *smart grid* benchmark focuses on predicting energy consumption loads to enable more efficient energy distribution. Measurements are generated from smart plugs installed in private households, and data is grouped by house. Two independent queries calculate the average energy consumption at local and global levels. To maintain reasonable throughput, we employ sliding windows with a 10 second duration and a 3 second sliding interval.

The results, shown in Table 9: ③, indicate the accuracy of ZeroTune models in predicting performance (latency and throughput) for these unseen benchmarks. The findings demonstrate that ZeroTune estimates both latency and throughput with high accuracy for unseen parallel query structures from public benchmarks. Overall, while both metrics are estimated ac-

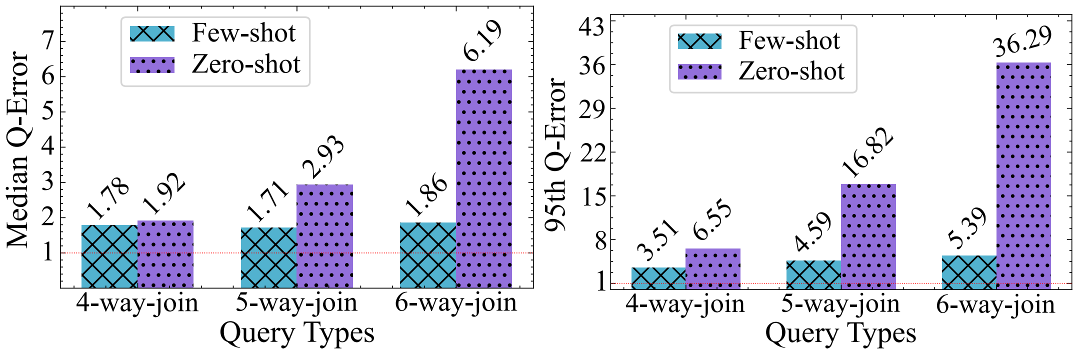


Figure 37: Few-shot learning with just 500 training queries for unseen 4-, 5-, and 6-way joins enhances ZeroTune's throughput prediction by nearly 6× for 6-way joins. The red line represents the perfect estimate [8].

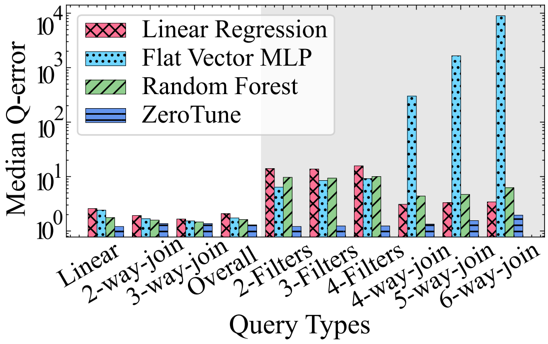


Figure 38: ZeroTune model provides robust and generalized cost predictions for both seen (white) and unseen (grey) PQP, outperforming other model architectures [8].

curately, latency predictions tend to be more precise than throughput predictions. The trend is consistent across all models and can be attributed to the nature of throughput, which is directly influenced by incoming data distribution. In contrast, latency is affected by indirect factors such as overall system utilization and the varying durations required to fill count windows.

Few-shot Learning. To further improve q-errors for complex, unseen PQP, we employ a machine learning technique known as *few-shot* learning. It involves training the model with a small number of additional examples to refine its predictions. Therefore, we train ZeroTune with an additional 500 examples of complex join structures. The results, illustrated in Figure 37, confirm that the model remains robust by adapting to the dynamic characteristics of DSP systems.

Few-shot learning improves accuracy with few queries

Comparison with Baselines. ZeroTune consistently outperforms baselines using flat vector representations, even when applies to unseen query structures. The superior performance is attributed to the parallel graph-encoding method, which effectively captures the complexities of query struc-

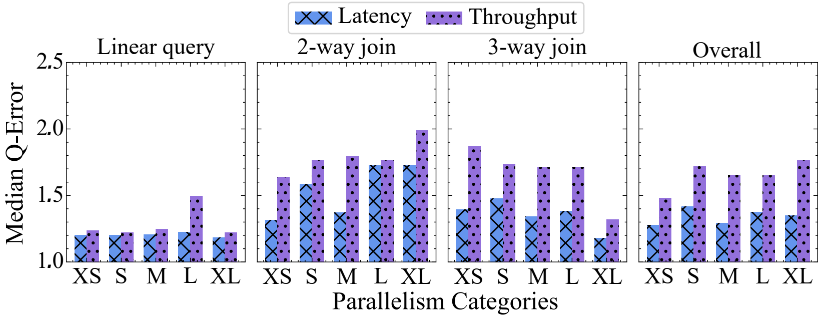


Figure 39: Median q-error of cost prediction of PQP for varying parallelism for seen query structures [8]. ZeroTune models trained using OptiSample show accurate cost prediction for seen PQP with varying parallelism categories.

tures. This capability enables ZeroTune to excel even with novel query configurations, as demonstrated in Figure 38.

6.3.2 *Fine-grained Parallelism Analysis*

Accuracy of model of increasing parallelism

Exp. 2: Impact of Parallelism. We further investigate the accuracy of ZeroTune across different degrees of parallelism, which is crucial for the optimizer (refer to Exp. 6) when selecting the optimal degree. Parallelism is classified into five categories: *XS*, *S*, *M*, *L*, and *XL*, each representing the average degree of parallelism per operator in a query.

For instance, in the *XL* category, each operator in a query uses an average of 64 to 128 cores (cf. Table 8). We assess how ZeroTune generalizes to both seen and unseen query types and public benchmarks within each parallelism category.

The results illustrate robust generalization capabilities of ZeroTune across all parallelism categories, maintaining high accuracy in its predictions for both *seen* and *unseen* query structures and public benchmarks. The analysis underscores the model's versatility and effectiveness in optimizing performance across varying degrees of parallelism.

Seen Workloads. In Figure 39, we present the accuracy of the ZeroTune model for seen query types within our test range (cf. Table 9 ①). The ZeroTune model consistently delivers highly accurate cost predictions across different parallelism degrees for these familiar query types.

Consistent accuracy for seen PQP...

While the complexity of the PQP can slightly impact prediction accuracy, the model performs overall exceptionally well. As parallelism and complexity increase towards the *XL* category, there is a minor decrease in accuracy.

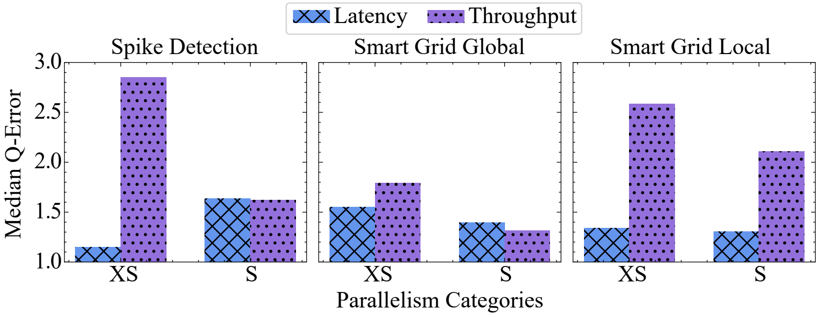


Figure 40: Median q-error of cost prediction of PQP for unseen query benchmark [8]. ZeroTune shows high accuracy for performance prediction for unseen benchmark PQP with different parallelism categories.

This reduction is due to the heightened computational overhead and data dependencies associated with more complex PQP.

...with increasing parallelism.

Despite this, the decline in accuracy is relatively small, demonstrating that ZeroTune maintains reasonable performance even for complex queries with high degrees of parallelism, highlighting the model's robustness and effectiveness in handling a variety of parallel query scenarios.

Unseen Benchmark. We extend our evaluations to analyze the accuracy and generalization of the ZeroTune model for unseen benchmark queries and to examine the impact of different parallelism categories. Our findings indicate that the OptiSample strategy often selects lower parallelism degrees, specifically XS and S categories, due to the simplicity and relatively low incoming event rates of the benchmarks, as illustrated in Figure 40.

ZeroTune *able transfer knowledge...*

The results demonstrate that the ZeroTune model accurately predicts costs across various parallelism categories, even in the context of benchmark queries. While the model performs well overall, it exhibits slightly higher prediction errors for throughput compared to latency. This discrepancy arises because the data distribution in benchmark queries differs from that of the synthetic queries, which aligns with the overall results (cf. Table 9 ③).

...to unseen PQP even of higher parallelism.

Despite these differences, fine-tuning the model can significantly enhance ZeroTune's performance for benchmark queries, which will be discussed in subsequent sections. The fine-tuning can help mitigate the impact of varying data distributions and further improve the accuracy of throughput predictions.

Unseen Resources. We also assess the ability of ZeroTune to generalize across *unseen* configurations of heterogeneous and homogeneous resources, as well as the impact of different parallelism categories (cf. to type "U" in Table 6). In Figure 41, the median q-errors, ranging from 1.25 to 2.0, indicate

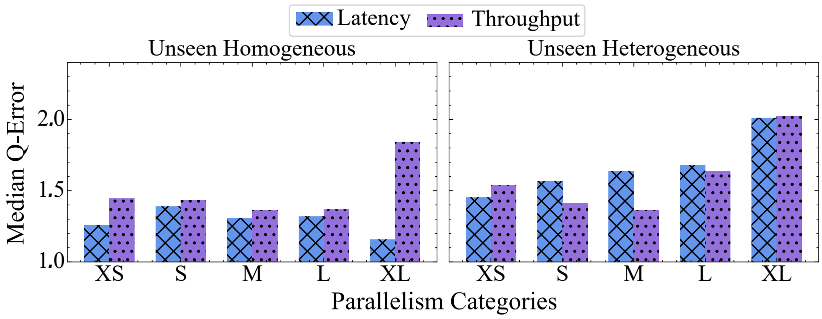


Figure 41: Median q-error of cost prediction of PQP for unseen homogeneous and heterogeneous resources [8]. ZeroTune shows high accuracy in performance prediction for PQP with different parallelism categories on unseen homogeneous and heterogeneous hardware configurations.

that the ZeroTune model consistently provides accurate cost predictions for *unseen* hardware resources and their associated parallelism categories.

Consistent performance across...

The results, shown in Figure 41, reveal that ZeroTune maintains accuracy even with *unseen* hardware setups. However, as with other scenarios, there is a noticeable increase in q-errors with higher degrees of parallelism. This trend can be attributed to complex factors such as load imbalance, the granularity of parallelism, and resource contention that become more pronounced with increased parallelism.

...diverse hardware resources.

Despite these challenges, the relatively low q-errors demonstrate the model's robust capability to learn from a variety of hardware configurations. The model effectively correlates these configurations with parallelism degrees, thanks to the transferable features related to resources that ZeroTune has learned. The adaptability ensures that the model remains reliable and accurate across diverse and unseen hardware environments.

High accuracy for unseen PQP...

Unseen Workload. In Figure 42, we evaluate the accuracy of ZeroTune for unseen and complex PQP across different parallelism categories. The results indicate that the ZeroTune model consistently delivers reasonably accurate predictions across all parallelism categories, with only minor variations. This consistency indicates that ZeroTune effectively captures performance patterns and generalizes them well to unseen or new query plans within the given template structure.

...even for higher parallelism.

However, we observe a slight decline in prediction accuracy, i.e., high q-errors for more complex PQP in comparison to simple PQP, particularly in throughput predictions for 5-way and 6-way joins, as indicated by the q-error trends in Table 9 ②. Throughput predictions for these complex plans were slightly less accurate, likely due to increased computational complexity and data dependencies.

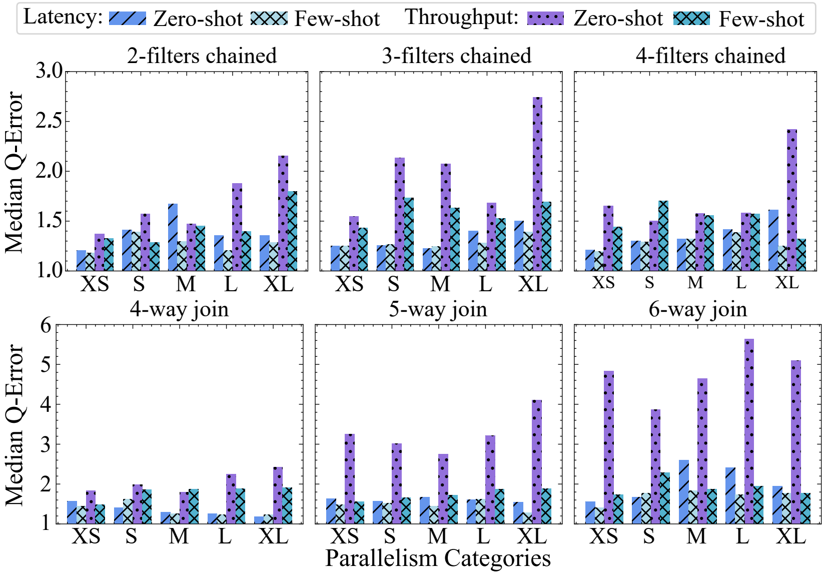


Figure 42: Median q-error of cost prediction of PQP for unseen query structures [8]. ZeroTune shows high accuracy in performance prediction for unseen PQP with different parallelism categories. Notably, the accuracy of ZeroTune decreases for highly complex unseen PQP with high parallelism categories. However, the model achieves accurate performance predictions when trained using few-shot learning with an additional 500 queries.

To mitigate this issue and enhance model performance, we fine-tune the model with an additional 500 queries. By employing a machine learning technique known as *few-shot learning*, as illustrated in Figure 42, we significantly improve prediction accuracy for both median and tail errors compared to *zero-shot learning*. The approach allows the model to adapt more effectively to each parallelism category, ranging from XS to XL, in line with the overall results presented in Figure 37.

High extrapolation results in increased q-error

The method proves particularly effective in enhancing ZeroTune's ability to handle complex query plans, thereby reducing the q-error and ensuring more precise throughput and latency predictions across all tested parallelism categories. This improvement demonstrates the robustness and adaptability of the ZeroTune model to unseen PQP. With a small amount of additional data, ZeroTune can effectively manage diverse workloads with increasing complexities of unseen query plans, ensuring higher accuracy and reliability in its predictions.

Few-shot leads to improved accuracy

6.3.3 *Generalization for Unseen Parameters*

Exp. 3: Impact of unseen parameters. In this section, we evaluate the ZeroTune model's ability to generalize and predict costs across various workload parameters. To thoroughly evaluate this, we use both interpolation and extrapolation within the ranges specified in Table 8.

Evaluation of unseen parameters

Figures 43 to 47 highlights the impact related to query and its placement, such as tuple width, event rate, window configurations, and the number of available workers in a cluster. These parameters are chosen due to their significant influence on parallelism degrees and associated costs. For instance, the event rate is particularly relevant because of its effect on backpressure, while window configurations are crucial due to the parallel processing of data streams that involve different key-based windows.

Model learns from parameters...

The evaluation involves using a model trained on *linear*, *2-way*, and *3-way* join query structures to test its predictive accuracy and generalizability across both interpolation and extrapolation ranges. This approach helps determine how well the model can adapt to variations within and beyond the training data.

...like backpressure from event rate.

To ensure a comprehensive evaluation of the model's performance, we conduct experiments using at least 165 queries per tuple width for each query, ensuring an equal distribution among *linear*, *2-way*, and *3-way* join queries within the parallel query structure. By maintaining the balance, we could effectively assess the ZeroTune model's capability to accurately predict costs under unseen conditions and varying degrees of parallelism.

By evaluating these parameters, we aim to understand how different factors influence the model predictions and validate its capability to handle diverse workload parameters, providing insights into its adaptability and accuracy in real-world applications. The evaluation highlights the model's robustness and adaptability in predicting performance metrics accurately across a wide range of configurations, ensuring its practical applicability.

Tuple Widths. To assess the accuracy of the ZeroTune model, we investigate the impact of different unseen tuple widths (cf. Figure 10), particularly in the context of higher degrees of parallelism. Specifically, we test the model with tuple widths ranging from 2 to 15 and unseen range is from 6 to 15 (cf. Tuple widths in Table 8).

ZeroTune *evaluation for unseen tuple widths*

In Figure 43, the model demonstrates stable performance and effective generalization to unseen tuple widths (indicated in the grey area). This stability suggests that the ZeroTune model has successfully learned the relationships between tuple width and associated costs in parallel query structures.

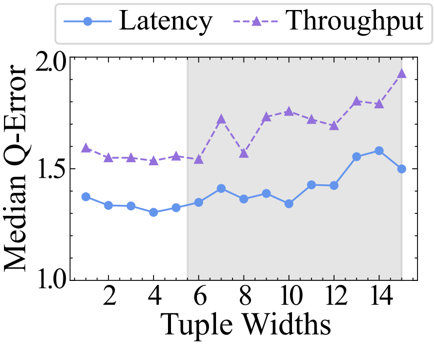


Figure 43: Median Q-errors for PQP cost prediction with varying tuple width (per data type of src.) where white shows training range and grey shows unseen range. ZeroTune accurately generalizes, even with higher extrapolation of tuple width [8].

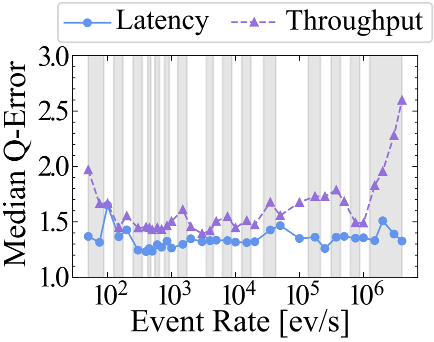


Figure 44: Median Q-errors for PQP cost prediction with varying event rates where white shows the training range, grey shows the unseen range. ZeroTune accurately generalizes, even with higher extrapolation of event rates [8].

As tuple width increases, the model continues to accurately predict costs, indicating its robust understanding to larger tuple widths that can influence computational demands and resource allocation.

ZeroTune generalizes across unseen tuple widths

The ability to generalize well across different tuple widths is crucial, as it shows that the model can adapt to different unseen data payload sizes in real-world applications. This adaptability ensures that ZeroTune can provide reliable cost predictions even when encountering tuple widths beyond those seen during training. The consistent performance of the model across both seen and unseen tuple widths highlights its capability to handle diverse data sources and optimize parallelism effectively. Overall, the results indicate that ZeroTune is well-equipped to manage and predict costs in environments with varying tuple widths, leveraging its learned correlations to maintain high accuracy and efficiency in cost estimation.

Event Rates. In Figure 44 (note a log scale on the x-axis), we evaluate the ZeroTune model's capability to interpolate and extrapolate event rates

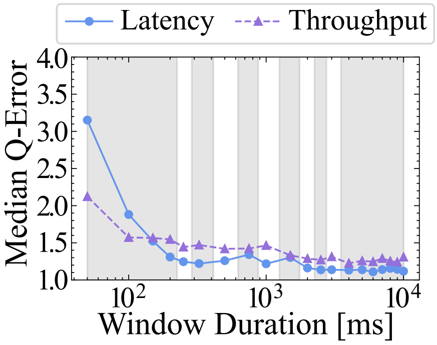


Figure 45: Median Q-errors for PQP cost prediction with varying window durations where white shows the training range, grey shows the unseen range. ZeroTune accurately generalizes, even with higher extrapolation of window duration [8].

beyond the training range (cf. Table 8). The model demonstrates high accuracy in predicting costs across a wide spectrum of event rates, including both low and high rates, within the training range (white area) and beyond the training range (shaded area).

High accuracy for varying event rates

The proficiency of the model in handling higher event rates can be attributed to its learned understanding of the processing limits of the DSP system and the backpressure effects that occur when the hardware is operating at full capacity. By effectively recognizing these limits, ZeroTune can accurately predict the costs associated with high event rates.

At very low event rates, however, there is a slight increase in q-error values. This is likely due to the model's limited exposure to scenarios of low utilization, where minimal data processing can lead to under-utilization of resources. In such cases, the system's behavior may differ significantly from more typical higher utilization scenarios, posing a challenge for accurate cost prediction.

Lower event show some deviation

Despite this, the overall performance of ZeroTune remains robust. The model demonstrates excellent generalization capabilities for both seen and unseen event rates, effectively predicting latency and throughput. This robustness is critical for adapting to various real-world scenarios where event rates can fluctuate widely.

Window Durations (Time-based). Figure 45 (note a log scale on the x-axis) illustrates the ZeroTune model's capability to accurately predict costs for PQP across a range of window durations (cf. Table 8). Initially, the model shows slightly higher median q-errors for smaller unseen windows. However, as the window duration increases, the performance of the model improves, ultimately converging to exhibit strong accuracy.

Accurate cost prediction...

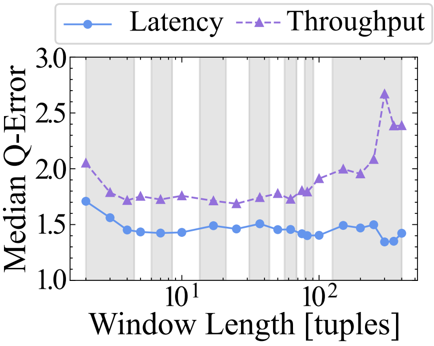


Figure 46: Median Q-errors for PQP cost prediction with varying window length where white shows the training range, grey shows the unseen range. ZeroTune accurately generalizes, even with higher extrapolation of window length [8].

The challenge with smaller windows arises because they lead to rapid data processing and high data turnover, making it difficult for the model to capture the performance characteristics influenced by parallelism. This is particularly challenging when the model has not been exposed to such small windows before, with the smallest seen window being 250 milliseconds. Rapid processing requires the model to quickly adapt to changing data patterns, which can be complex without prior training on smaller windows.

...and generalization...

Conversely, longer windows allow for more data accumulation before processing, which helps the model to better understand system patterns and thus achieve higher accuracy in cost estimation. This is because longer windows provide a more stable environment, reducing the variability and allowing the model to make more informed predictions. However, as window durations extend towards the far end of the unseen range, there are slight variations in accuracy. This could be attributed to the fewer training examples available for extremely long windows, which limits the model's ability to generalize as effectively.

...for small and big windows durations.

Overall, the ZeroTune model generalizes particularly well for longer windows, offering highly accurate predictions. For smaller windows, although the initial predictions are less accurate, the model still provides reasonably good predictions as it adjusts to the rapid processing demands. This demonstrates the model's robustness and flexibility in handling a wide range of window durations, ensuring reliable performance across various data processing scenarios.

Window Lengths (Count-based). Similar to window duration, window length also significantly affects the cost of PQP, with varying impacts on throughput depending on whether the windows are *time-based* or *count-based*.

Window length influences throughput

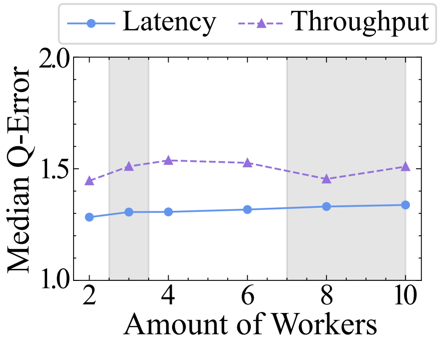


Figure 47: Median Q-errors for PQP cost prediction with varying amount of workers. White indicates the training range, and grey indicates the unseen range. ZeroTune demonstrates accurate generalization, even with higher worker extrapolation [8].

In the case of *time-based* windows, throughput tends to remain constant, as it is generally unaffected by the rate of incoming events. This stability simplifies throughput prediction. On the other hand, *count-based* windows have a throughput that is directly influenced by the input rate of the operator. For example, a tumbling count-based window with a length of 10 will reduce the outgoing event rate to 10% of the incoming event rate. This relationship makes predicting throughput more challenging for unseen window sizes in count-based scenarios.

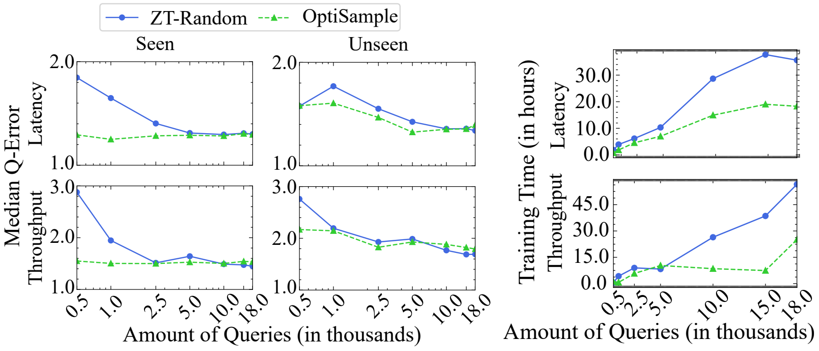
Overall high accuracy for varying window lengths

As shown in Figure 46 (note the log scale on the x-axis), the ZeroTune model demonstrates strong accuracy and generalization in predicting costs for both seen and unseen window lengths. The model's performance in this regard highlights its ability to adapt to different window configurations. However, there is a slight increase in q-error for throughput predictions when dealing with extremely short or long unseen window lengths. This increase is due to the inherent complexities associated with accurately forecasting throughput for these extreme window sizes, as previously mentioned. Despite this, the overall accuracy remains high, demonstrating the model's robust capability to handle a wide range of window lengths effectively.

These findings underscore the importance of considering window length as a critical parameter in PQP cost prediction. The ZeroTune model's proficiency in this area ensures that it can provide reliable predictions across diverse window configurations, thereby optimizing performance and resource allocation in streaming data environments.

Computation resources influence possible parallelism

Amount of Workers. The number of workers or nodes in a DSP system significantly impacts the overall cost, as it directly determines the available computational resources, the degree of parallelism, and the system's capacity to manage workloads. For instance, smaller clusters with fewer workers



(a) Amount of queries

(b) Training time

Figure 48: (a) Median Q-errors for models trained with different quantities of seen (left) and unseen (middle) queries. (b) Training times necessary for ZeroTune to reach efficiency using both random (ZT-Random) and OptiSample-based data. The OptiSample-based strategy achieves equivalent accuracy with half the data and training time [8, 6].

may experience limited parallelism due to the reduced number of available computational units. Conversely, larger clusters can support a higher number of parallel tasks, thereby enhancing the system's ability to process more complex and extensive workloads.

In our evaluation, depicted in Figure 47, we demonstrate that the ZeroTune model consistently delivers accurate cost predictions across various cluster sizes. This capability enables the model to effectively generate PQP that scale with the increasing size of the cluster. As the number of workers grows, the model adjusts the level of parallelism accordingly, ensuring efficient resource utilization and optimized performance.

Higher generalization with increasing number of workers

These results show the model's robustness and flexibility in adapting to different computational environments. By accurately predicting costs and scaling parallelism based on cluster size, ZeroTune proves to be a reliable tool for optimizing performance in dynamic and heterogeneous DSP systems. This adaptability ensures that whether dealing with small-scale or large-scale clusters, the system can maintain high efficiency and handle workloads effectively.

6.3.4 *Data-efficient Training*

Amount of PQP influence accuracy

Exp. 4: Impact of training strategy. In Figure 48a, we evaluate the accuracy of our ZeroTune models by comparing two training strategies: the random data approach (ZT-Random) and the OptiSample strategy, plotted against the increasing number of queries (note the log scale on the x-axis). For both seen and unseen data, the OptiSample-based model shows rapid convergence, achieving high accuracy with as few as $5,000$ queries. In contrast, the random model requires over $18,000$ queries to reach a similar level of accuracy. This demonstrates the superior data efficiency of the OptiSample strategy.

OptiSample achieves high accuracy...

Further analysis in Figure 48b (note a linear scale on the x-axis) examines the training time required as the number of queries increases. The OptiSample model achieves higher accuracy in roughly half the time, taking approximately 4.6 hours, compared to the random model, which takes about 10.3 hours. This finding supports our hypothesis that the OptiSample strategy is more data-efficient, achieving equal or better accuracy with fewer queries and significantly less training time.

...with fewer queries and half of the training time.

These results address a critical bottleneck in training zero-shot models (cf. C3: high training effort for generalization in Section 5.2), where the challenge is to obtain high accuracy without extensive data or prolonged training periods. The OptiSample strategy's ability to converge faster with fewer queries and reduced training time shows its effectiveness in optimizing the training process. This efficiency not only accelerates model development but also enhances the practical applicability of the ZeroTune models in dynamic and data-intensive environments.

Overall, the comparative analysis validates that the OptiSample strategy significantly outperforms the random data approach, making it a valuable method for training efficient and accurate zero-shot models.

6.3.5 *Optimizer for Parallelism Tuning*

ZeroTune tunes initial parallelism...

Exp. 5: Impact of ZeroTune in parallelism determination. We evaluate how ZeroTune performs in selecting optimal parallelism degrees together with the optimizer, as detailed in Section 5.4.3. In Figure 49a, we present the mean speed-ups for various query structures, including those that are previously unseen. The metric indicates the speed-up factor achieved by executing queries using the parallelism degrees selected by the ZeroTune model, compared to a greedy heuristic approach.

...faster than baselines...

For this evaluation, we randomly selected 100 query types with different parameters, ensuring they remained deterministic. We then perform inference for different parallelism degrees based on the enumeration strategies. The results show that ZeroTune significantly outperforms the greedy heuristic approaches [83, 225], achieving speed-ups of up to 12 times for simple linear queries and approximately 3.04 times for unseen and more complex query types, such as n-way joins.

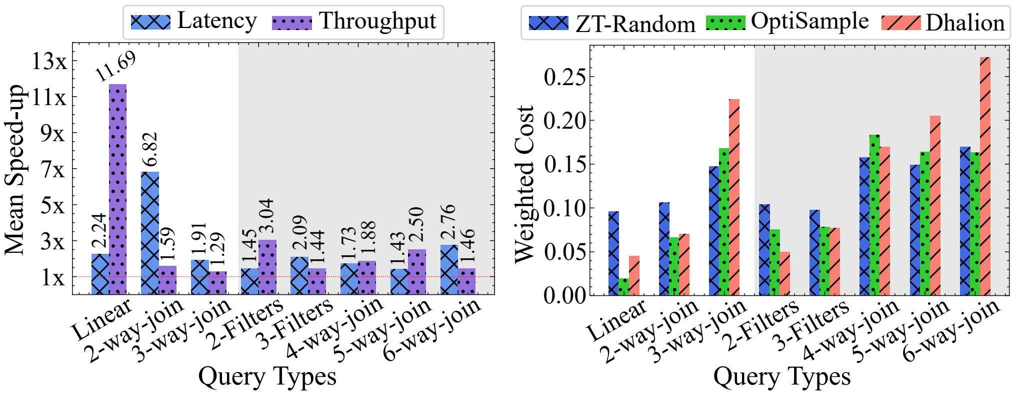


Figure 49: (a) Mean speed-up in latency and throughput achieved through parallelism tuning by ZeroTune compared to greedy heuristic approaches [83, 225]. (b) Comparative performance of query types with parallelism degrees determined by ZeroTune versus *Dhalion* [83], showing weighted average runtimes. ZeroTune outperforms *Dhalion* in both seen and unseen queries [8].

Additionally, we compared the performance of the proposed model with *Dhalion* [83], a well-known traditional (non-learned) parallelism tuning algorithm used in the Heron DSP system [144]. *Dhalion* is a state-of-the-art auto-scaling controller widely accepted in both academia and industry, particularly at Twitter with their Heron system, making it a suitable benchmark for comparison.

The promising results of ZeroTune, especially its superior performance over non-transferable representations like flat-vector in previous experiments, motivated us to compare it against these well-established auto-scalers. The comparison highlighted the effectiveness of ZeroTune in optimizing parallelism degrees, demonstrating its potential to enhance query execution performance significantly.

...like Dhalion for Heron DSP system.

Overall, the ZeroTune model's ability to select optimal parallelism degrees leads to substantial speed-ups, proving its value in improving the efficiency and performance of DSP systems. This makes ZeroTune a learned system for real-time data processing applications, ensuring high performance even with complex and unseen query structures.

The results depicted in Figure 49b, which show the weighted average cost (cf. Equation (12)), reveal important insights into the performance of different parallelism tuning strategies. While *Dhalion* performs relatively well for similar and straightforward query structures—aligning with its design focus—its effectiveness diminishes as the complexity of parallel queries increases.

ZeroTune finds optimal parallelism...

In contrast, our ZeroTune model consistently identifies cost-effective parallelism degrees across a wide range of queries, including both familiar and

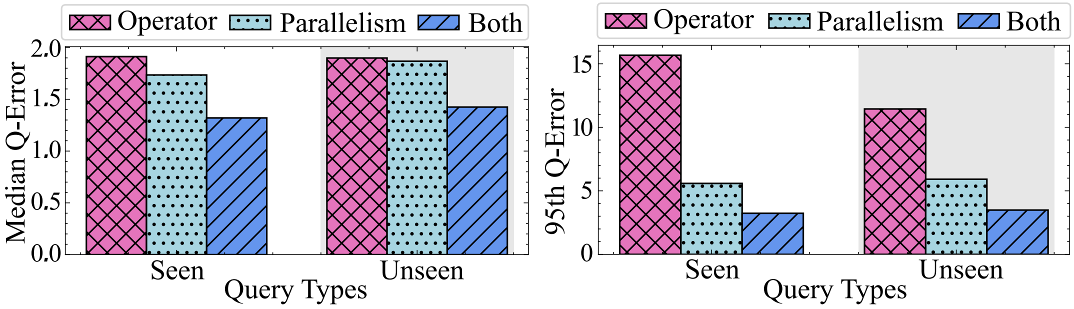


Figure 50: Feature ablation analysis [8] using (1) only operator-specific features, (2) only parallelism- and resource-related features, and (3) all combined features for latency cost prediction.

novel structures. This robust performance is evident regardless of the query complexity. Unlike *Dhalion*, which struggles with more complex PQP, ZeroTune maintains high efficiency and accuracy.

...using predicted cost...

The superior performance of ZeroTune is attributable to its advanced learning algorithms that enable it to generalize well from training data, thereby effectively managing diverse and complex query environments. This capability ensures that ZeroTune not only excels in optimal resource allocation for simple queries but also adapts seamlessly to handle the intricacies of more complex parallel queries.

...without executing query.

By consistently outperforming *Dhalion*'s traditional tuning algorithm, ZeroTune shows its potential as a performance prediction model for optimizing parallelism in DSP systems, leading to significant cost savings and performance enhancements, making it valuable for dynamic and heterogeneous data processing applications where query complexity can vary widely.

6.3.6 *Ablation Study*

Each feature plays a vital role

Exp. 6: Impact of different transferable features. We conduct a feature ablation study to identify and quantify the contribution of each transferable feature to the model's generalization performance. The analysis aims to understand how different combinations of feature sets impact the prediction accuracy of the zero-shot cost model for PQP. Figure 50 illustrates that *operator-specific* features, such as average selectivity, enhance the model's comprehension of data processing tasks. However, these features alone do not significantly boost the accuracy of predictions for PQP. While these features provide valuable insights, they are insufficient by themselves to fully capture the complexities of parallel query execution.

In contrast, when we integrate *operator-specific* features with *parallelism-specific* features, such as degrees of parallelism, the performance of the model improves markedly. The combination allows the model to correlate data processing characteristics with the efficiency of parallel execution and resource utilization. By bridging these aspects, the model can more effectively predict the costs associated with both seen and unseen PQP. The integration of *parallelism-specific* features enhances the model's ability to generalize across different query scenarios. This improved performance is evident in the model's ability to accurately predict costs, reflecting its efficiency in managing diverse and complex data processing environments. The results of this feature ablation study underscore the importance of a holistic feature set that encompasses both data processing and parallel execution characteristics.

Combined features improve the accuracy

Overall, this analysis highlights the critical role that a well-rounded set of features plays in enhancing the predictive accuracy and generalization capabilities of the ZeroTune model. By effectively leveraging a combination of operator-specific and parallelism-specific features, the model demonstrates robust performance in predicting costs for a wide range of PQP.

6.4 Summary

In this chapter, we present evaluation results to highlight the capabilities of proposed performance modeling methods *(i)* the performance benchmarking capabilities of PDSP-Bench and *(ii)* accurate performance prediction and generalization capabilities of ZeroTune to predict performance for parallel and distributed stream processing.

Initially, we evaluate in real-world distributed stream processing setup, which includes CloudLab testbed for distributed infrastructure and well-known DSP system, i.e., Apache Flink as SUT for performance benchmarking and workload generation under varying resource configurations and query parameters. Subsequently, we provide a view of evaluation metrics for performance benchmarking and measuring performance prediction accuracy and introduce a possible baseline for comparison.

Later, we provide performance benchmarking results of PDSP-Bench, explicitly focusing on end-to-end latency and the impact of workload and hardware diversity on performance. Consequently, we present interesting observations such as *non-linearity* effect of parallelism and *diversity dilemma* of heterogeneous hardware in enhancing performance.

Finally, we also provide performance prediction results of ZeroTune for both end-to-end latency and throughput for seen-unseen workloads, across parallelism degrees, unseen operator parameters, and resource configurations. Additionally, we show the generalization capability of ZeroTune and efficiency of the data-efficient training method OptiSample compared to naive approaches. Following up, we present the capability of optimizer in combination with ZeroTune to provide optimal initial parallelism degree based on predicted performance and show speed-ups achieved with the parallelism tuning approach in comparison to other approaches.

7

Summary, Conclusion and Outlook

This thesis contributes methods and models that solve the research challenges *RC1*: understanding the performance and *RC2*This thesis contributes methods and models that solve the research challenges *RC1*: understanding the performance and *RC2*: accurate performance prediction for parallel and distributed stream processing (PDSP). In the following, we summarize the content of previous chapters followed by a brief revisit of the two key contributions and obtained results in Section 7.1 Finally, we provide an outlook on potential directions for future work in Section 7.2.

7.1 Summary of the Thesis

Overall summary of thesis

In Chapter 1, we discussed the increasing processing demands of modern data-driven applications, emphasizing the importance of *timeliness* and *throughput* requirements. We remarked on the role of Distributed Stream Processing (DSP) systems in meeting these demands, particularly through the use of heterogeneous cloud environments. We highlighted the necessity of accurate performance modeling in DSP systems operating in heterogeneous environments and outlined the challenges of developing such models. Specifically, we identified two key challenges, *RC1:* understanding the performance and *RC2:* performance prediction and optimization for DSP workloads in heterogeneous cloud environments. Based on our findings and analysis of the state of the art, we defined two main research goals that we address in this thesis: *RG1* systematic performance benchmarking of PDSP workloads in a heterogeneous environment and *RG2* generalizable and data-efficient learned performance models for DSP in a heterogeneous environment. In Chapter 2, we provided essential background information on DSP and reviewed existing research on performance modeling for DSP systems. This chapter identified the limitations and pitfalls in addressing the research challenges mentioned in Chapter 1. In Chapter 3, we presented the overall architecture, system model, and scenario for the accurate performance modeling problem. Here, we presented the overall scenario and highlighted the importance of accurate performance prediction for optimizing DSP systems, such as operator placement and parallelism. Additionally, we outlined the overall system

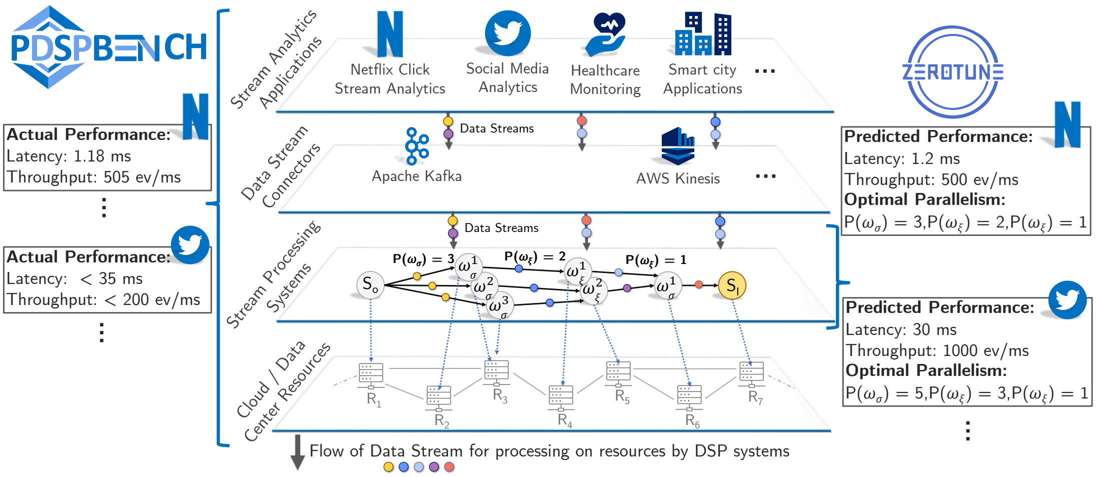


Figure 51: Summary of all the contributions in a single architecture for PDSP. The proposed contributions in this thesis (1) PDSP-BENCH [7]: A benchmarking system for parallel and distributed dataflow of DSP applications for *RC1* and (2) ZEROTUNE [8]: a zero-shot cost model to predict accurate performance of parallel dataflow applications even when they execute on heterogeneous hardware to tackle *RC2*.

model that outlined the important assumptions and system entities for performance benchmarking and prediction. We summarized the contributions (cf. Figure 51) that covered the individual research goals and the results of this thesis, as discussed in the following.

7.1.1 *Contributions Revisited*

PDSP-BENCH: *a novel benchmarking system...*

...for parallel dataflows...

...in heterogeneous cloud environments.

In Chapter 4, we proposed a performance benchmarking system that enables an enhanced understanding of the performance of executing PDSP workloads in a heterogeneous cloud environment. We discussed the limitations of existing performance benchmarking work in addressing the requirements for benchmarking PDSP workloads and identified three primary challenges for our work: lack of expressiveness in benchmarking PDSP workloads, the necessity for heterogeneous hardware support, and the need to benchmark learned DSP models by their integration into the system. Thus, the PDSP-BENCH system [7] successfully addressed the first research goal, i.e., performance understanding of PDSP workloads in a heterogeneous cloud environment by offering the following contributions.

(i) With PDSP-BENCH, we enabled the evaluation of parallel query structures (PQP) across a diverse range of DSP synthetic and real-world workloads on heterogeneous cloud environment, thus offering both an expressive and scalable solution (cf. Section 4.3).

(ii) We provided methods to configure and manage heterogeneous hardware resources by integrating resources from research testbeds like CloudLab [76] for accurately reflecting real-world deployment scenarios (cf. Section 4.5).
(iii) We facilitated the integration of learned DSP models, allowing for systematic training and evaluation of these models on diverse streaming workloads, which is essential given the surge of use of ML for optimizing DSP performance [8], [106], [265].
(iv) Lastly, we enabled the generation of large corpora of streaming datasets via PDSP-BENCH to ensure that the ML models are trained on representative data of actual real-world streaming workloads.

In Chapter 5, we proposed performance prediction methods and models for DSP mechanism optimization and enabling transition in the diverse and heterogeneous environments induced by DSP and cloud resources. Following this, we provided the performance prediction scenario and reviewed existing work to identify the challenges in performance prediction and optimization in the context of the second research goal. We identified three main challenges of existing work: lack of generalization in workload-driven learning methods, high training effort for generalization, and inaccurate performance modeling, leading to incorrect initial provisioning of parallel operators. We proposed ZEROTUNE, which successfully achieved the second research goal by solving these challenges with the following contributions.

ZEROTUNE*: a novel performance prediction model...*

...for parallel dataflows.

(i) We presented ZEROTUNE, a novel zero-shot cost model designed to predict execution costs of parallel DSP queries and thereby initially configuring parallelism degrees while avoiding costly trial-and-error adjustments from the start of the query execution.
We employed a novel learning paradigm called data-efficient zero-shot learning [46, 222], enabling ZEROTUNE to grasp the dynamics of a DSP system offline. It allowed the model to apply learned insights across various DSP workloads in less training effort.
(ii) We achieved generalization in cost predictions using ZEROTUNE by offering novel *transferable* feature attributes (cf. Section 5.3) and *parallel graph representation* for training the neural network (cf. Section 5.2).
(iii) We encoded the transferable features and parallel graph representation in the GNN model architecture, which learned the query costs by message passing algorithms (cf. Section 5.4).
(iv) We introduced OptiSample, a data-efficient training strategy that leverages analytical methods to estimate and explore meaningful parallelism degrees, collecting meaningful data for training the GNN model.
(v) Lastly, we combined inference with an optimizer that uses predicted costs from the ZEROTUNE model to provide an initially optimized set of parallelism degrees for each operator without executing the query.
Finally, we performed an extensive evaluation of both contributions in Chapter 6 to show the capabilities and applicability in real-world scenarios under

ZEROTUNE *enables generalization...*

...across unseen hardware and workloads.

varying workloads and resource configurations of the cloud environment, as summarized in the next section.

7.1.2 *Conclusions on Key Results*

PDSP-Bench *enables benchmarking parallel dataflows...*

In Chapter 6, we evaluated our contributions to understand their capabilities for performance benchmarking of parallel dataflows and heterogeneous configurations, data generation, performance prediction, and optimization, as well as the generalization capabilities of the proposed model to both unseen workloads and hardware of the cloud environment. In Section 6.2, we benchmarked the performance of a well-known DSP system, Apache Flink, for diverse workloads and hardware to understand the influence of parallelism on performance. We showed that PDSP-Bench can benchmark performance for a broad spectrum of parameters as specified in Section 6.2: Table 7.

...and reveals insights on non-linear effects...

...and diversity dilemma of hardware.

We identified 8 key observations (O1-O8) found based on our benchmarking system PDSP-Bench related to workload and hardware diversity on the performance of a DSP system. The observations highlight the impact of increasing parallelism on the performance, such as the *non-linear* effects on the performance metrics such as latency of a parallel dataflow. Another key observation is the *diversity dilemma* of hardware, as there is no clear choice of homogeneous or heterogeneous clusters for all parallel dataflows. Finally, the analysis of the learned cost models revealed the observation that graph representation assists in learning the performance of parallel dataflows in a DSP system, on which we base our second contribution.

ZeroTune *enables performance prediction...*

...and shows high prediction accuracies...

...with speed-ups of upto 5× in optimizing parallelism degrees.

In the second evaluation in Section 6.3, we showed the prediction accuracy and generalization capability of ZeroTune, the zero-shot cost models for performance prediction of parallel DSP queries. ZeroTune showed very high prediction accuracies and generalization abilities of ZeroTune for different scenarios, including seen and unseen workloads, different parallelism degrees, and unseen operator parameters (cf. Table 8). For some cases where the prediction accuracy dropped due to high expectations of the model to extrapolate much higher, e.g., for 6-way join workload, we showed that fine-tuning the model with as little as 500 queries improved the accuracy by almost 6× and enabled precise predictions for significantly different workloads. In the optimization performance, our experiments showed that ZeroTune enabled on average a speed-up of about 5× in selecting optimal parallelism degrees for operators of parallel dataflows compared to baseline approaches and therefore significantly outperformed existing analytical baselines [83, 90, 225]. Furthermore, by training ZeroTune using our novel data-efficient training strategy OptiSample, we significantly reduced the training effort by 4× for generalization.

In summary, our contributions, PDSP-Bench and ZeroTune, provide significant advancements in the field of DSP systems. They offer robust benchmarking and performance prediction methods, facilitating the optimization of PDSP in heterogeneous environments. This work not only enhances the understanding of parallel and DSP performance but also paves the way for more efficient and adaptive DSP systems in the future.

7.2 Future Outlook

The results presented in this thesis provide the foundation for further research in DSP. A notable achievement is that recently, our approach of performance prediction has been adopted by *Amazon Redshift* for query execution time prediction [258] and is being further developed. We provide an overview of possible directions where the proposed methods can be applied and extended to open problems.

Amazon Redshift adopted our approach

Heterogeneous hardware architectures

PDSP-Bench provides a platform for benchmarking the performance of comprehensive suites of applications, heterogeneous hardware configurations, and DSP mechanisms, specifically focusing on parallel processing. It can be easily extended to support benchmarking of different DSP systems, such as Storm [236] and Heron [144], as well as additional optimization mechanisms. However, the platform is currently limited to CPU architectures, as current DSP systems do not support other architectures like GPUs, FPGAs, etc. Thus, a promising direction for further exploration is the inclusion of other architectures such as GPUs, FPGAs, or even smart switches like P4 [139, 268, 270, 272, 257, 147, 43, 146, 31, 100].

PDSP-Bench supports heterogeneous configurations

This task is particularly challenging due to the lack of streaming systems that enable the support of such different hardware architectures. For instance, DSP system - *NebulaStream* [266] supports other architectures, but it is limited to IoT-specific devices. Therefore, an open research question remains: how to design a unified benchmarking system capable of performing comprehensive performance evaluations across various hardware architectures while including heterogeneous environments. For instance, each of these architectures has unique characteristics and performance metrics, making standardization on performance across architectures difficult, and it is even harder that they seamlessly work together in a distributed environment. This can result in research challenges like efficiently managing resources across different hardware architectures to ensure optimal performance, which includes dynamic resource allocation, load balancing, and

Next step to support heterogeneous architectures like GPUs scheduling. Notably, this requires developing benchmarks that scale across various hardware configurations, from small IoT devices to large clusters of GPUs, without losing accuracy or relevance.

Dynamic adaptation and online zero-shot learning

ZeroTune can be fine-tuned for other metrics and optimization

ZeroTune cost models offer significant advantages, such as data-efficient methods for accurate cost predictions and generalization to unseen workloads and resource configurations. They can be adapted with minor modifications, e.g., by fine-tuning to solve additional tasks, such as predicting additional cost metrics, e.g., resource utilization or optimizing joint parallelism and placement [253]. However, these models face challenges when dealing with large search spaces and online learning for dynamic optimization like parallelism adaptation [200, 4, 264] or re-scaling decisions [201, 54]. Therefore, one of the open research directions is how to design and develop online machine learning models for accurate optimization with generalization in a dynamic and heterogeneous environment, such as on-the-fly parallelism adaptation or resource re-scaling.

Next step can be end-to-end learning...

Unlike traditional approaches that navigate huge search spaces with heuristics and find an optimum based on a zero-shot cost model, an online zero-shot model would learn to navigate the search space itself, directly arriving at a solution without needing additional optimization procedures. For instance, integrating reinforcement learning (RL) [54, 201] with zero-shot models like ZeroTune can enhance their ability to make dynamic adaptation decisions. An RL agent can learn optimal scaling and parallelism adjustments by interacting with the environment, continuously improving its performance predictions based on real-time feedback. This approach can help address the limitations of current models and provide a more robust solution for dynamic adaptation in DSP systems.

...for dynamic adaptation.

Towards resource-efficient and Green-AI

ZeroTune support data-efficient learning

Next step can be resource-efficiency

In recent years, there has been a growing emphasis on the environmental impact of large-scale data processing and machine learning (ML) systems [165, 256]. ML for DSP systems, which are fundamental to real-time data analytics in various domains, consume significant computational resources, contributing to high energy usage and carbon footprints [109, 188, 19, 167, 166]. As data volumes continue to grow, the demand for more resource-efficient and environmentally friendly DSP solutions becomes increasingly critical. The motivation for resource-efficient and Green-AI [208] in DSP lies in the need to balance performance with sustainability, reducing the ecological footprint of ML while maintaining high levels of efficiency and accuracy.

ZeroTune has already made a contribution in this direction by proposing data-efficient training and maintaining the same level of accuracy in comparison to data-agnostic ML models. In the next step, one of the open research questions can be: how to design both data- and resource-efficient methods to reduce the energy footprint of training and inference phases in zero-shot models used for DSP? Investigating lightweight models [26, 210, 161], pruning techniques [246, 127, 261], and energy-efficient hardware accelerators [241, 227, 22] for model training and inference can contribute to sustainable prediction models and methods. For instance, lightweight models, which are designed to be smaller and more efficient, require less computational power and memory. This directly translates to lower energy consumption during both training and inference phases. Similarly, model pruning involves removing redundant or less important neurons and connections from the neural network. This not only reduces the size of the model but also decreases the energy required for computations.

Green-AI is essential...

...for environment-friendly AI solutions.

Acknowledgements

This work has been co-funded by the German Research Foundation (DFG) as part of the project C2 within the Collaborative Research Center (CRC) 1053 – MAKI.

Bibliography

[1] Daniel J Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag Maskey, Alex Rasin, Esther Ryvkina, et al. "The design of the borealis stream processing engine." In: *Proceedings of the Conference on Innovative Data Systems Research*. 2005, pp. 277–289 (cit. on p. 19).

[2] Zahraa S Abdallah, Mohamed Medhat Gaber, Bala Srinivasan, and Shonali Krishnaswamy. "Activity recognition with evolving data streams: A review." In: *ACM Computing Surveys (CSUR)* 51.4 (2018), pp. 1–36 (cit. on p. 1).

[3] Robert Adolf, Saketh Rama, Brandon Reagen, Gu-Yeon Wei, and David Brooks. "Fathom: Reference workloads for modern deep learning methods." In: *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*. 2016, pp. 1–10 (cit. on p. 17).

[4] Pratyush Agnihotri. "Autonomous Resource Management in Distributed Stream Processing Systems." In: *Proceedings of the 22nd International Middleware Conference: Doctoral Symposium*. 2021, pp. 19–22 (cit. on pp. 9, 10, 12, 13, 16, 19, 134).

[5] Pratyush Agnihotri, Boris Koldehofe, Carsten Binnig, and Manisha Luthra. "PANDA: Performance Prediction for Parallel and Dynamic Stream Processing." In: *Proceedings of the 16th ACM International Conference on Distributed and Event-Based Systems*. 2022, pp. 180–181 (cit. on pp. 16, 19, 22, 25, 76, 101).

[6] Pratyush Agnihotri, Boris Koldehofe, Carsten Binnig, and Manisha Luthra. "Zero-Shot Cost Models for Parallel Stream Processing." In: *Proceedings of the Sixth International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. 2023, pp. 1–5 (cit. on pp. 22, 23, 25, 76, 101, 111, 123).

[7] Pratyush Agnihotri, Boris Koldehofe, Roman Heinrich, Carsten Binnig, and Manisha Luthra. "PDSP-Bench: A Benchmarking System for Parallel and Distributed Stream Processing." In: *Proceedings of the Performance Evaluation and Benchmarking - 16th TPC Technology Conference, TPCTC*. 2024, pp. 1–22 (cit. on pp. 5, 6, 46, 97, 99, 104, 106, 130).

[8] Pratyush Agnihotri, Boris Koldehofe, Paul Stiegele, Roman Heinrich, Carsten Binnig, and Manisha Luthra. "ZeroTune: Learned Zero-Shot Cost Model for Parallelism Tuning in Stream Processing." In: *Proceedings of IEEE 40th International Conference on Data Engineering (ICDE)*. 2024, pp. 2040–2053 (cit. on pp. 5, 7, 12, 13, 16, 19, 23, 25, 33, 37, 45, 46, 55, 75, 76, 79, 80, 83–86, 97, 101, 108, 109, 111, 113–117, 119–123, 125, 126, 130, 131).

[9] Pratyush Agnihotri, Florian Weber, and Sascha Peters. "Axxessity: City and User-Centric Approach to Build Smart City Bonn." In: *Proceedings of the 13th ACM International Conference on Distributed and Event-Based Systems*. 2019, pp. 220–223 (cit. on p. 33).

[10] ACCA Think Ahead. *The unassailable rise of Netflix*. `https://www.accaglobal.com/gb/en/student/sa/features/netflix.html`. [Online; accessed 25-04-2024]. 2021 (cit. on p. 28).

[11] Nasim Ahmed, Andre LC Barczak, Mohammad A Rashid, and Teo Susnjak. “Runtime prediction of big data jobs: performance comparison of machine learning algorithms and analytical models.” In: *Journal of Big Data* 9.1 (2022), p. 67 (cit. on p. 17).

[12] Adnan Akbar, Francois Carrez, Klaus Moessner, Juan Sancho, and Juan Rico. “Context-aware stream processing for distributed IoT applications.” In: *Proceedings of the IEEE 2nd World Forum on Internet of Things (WF-IoT)*. 2015, pp. 663–668 (cit. on p. 23).

[13] Barş Akgün and şule Gündüz Öğüdücü. “Streaming linear regression on spark MLlib and MOA.” In: *Proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. 2015, pp. 1244–1247 (cit. on pp. 20, 75).

[14] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. “Understanding of a convolutional neural network.” In: *Proceedings of the international conference on engineering and technology (ICET)*. 2017, pp. 1–6 (cit. on p. 21).

[15] Bastian Alt, Markus Weckesser, Christian Becker, Matthias Hollick, Sounak Kar, Anja Klein, Robin Klose, Roland Kluge, Heinz Koeppl, Boris Koldehofe, Wasiur R. KhudaBukhsh, Manisha Luthra, Mahdi Mousavi, Max Mühlhäuser, Martin Pfannemüller, Amr Rizk, Andy Schürr, and Ralf Steinmetz. “Transitions: A Protocol-Independent View of the Future Internet.” In: *Proc. IEEE* 107.4 (2019), pp. 835–846 (cit. on p. 2).

[16] Flor Álvarez, Lars Almon, Patrick Lieser, Tobias Meuser, Yannick Dylla, Björn Richerzhagen, Matthias Hollick, and Ralf Steinmetz. “Conducting a Large-scale Field Test of a Smartphone-based Communication Network for Emergency Response.” In: *Proceedings of the 13th Workshop on Challenged Networks*. 2018, pp. 3–10 (cit. on p. 30).

[17] Henrique Andrade, Bugra Gedik, Kun-Lung Wu, and S Yu Philip. “Scale-up strategies for processing high-rate data streams in System S.” In: *Proceedings of the IEEE 25th International Conference on Data Engineering*. 2009, pp. 1375–1378 (cit. on p. 61).

[18] Leonardo Aniello, Roberto Baldoni, and Leonardo Querzoni. “Adaptive online scheduling in storm.” In: *Proceedings of the 7th ACM international conference on Distributed event-based systems*. 2013, pp. 207–218 (cit. on p. 65).

[19] Lasse F Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. “Carbontracker: Tracking and predicting the carbon footprint of training deep learning models.” In: *arXiv preprint arXiv:2007.03051* (2020) (cit. on p. 134).

[20] Arvind Arasu, Mitch Cherniack, Eduardo Galvez, David Maier, Anurag S Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. “Linear road: a stream data management benchmark.” In: *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. 2004, pp. 480–491 (cit. on pp. 2, 16, 44, 45, 56, 57, 62, 102).

[21] P Araujo, G Astray, JA Ferrerio-Lage, JC Mejuto, JA Rodriguez-Suarez, and B Soto. “Multilayer perceptron neural network for flow prediction.” In: *Journal of Environmental Monitoring* 13.1 (2011), pp. 35–41 (cit. on pp. 21, 102).

[22] Giorgos Armeniakos, Georgios Zervakis, Dimitrios Soudris, and Jörg Henkel. “Hardware approximate techniques for deep neural network accelerators: A survey.” In: *ACM Computing Surveys* 55.4 (2022), pp. 1–36 (cit. on p. 135).

[23] Alexander Artikis, Matthias Weidlich, Francois Schnitzler, Ioannis Boutsis, Thomas Liebig, Nico Piatkowski, Christian Bockermann, Katharina Morik, Vana Kalogeraki, Jakub Marecek, et al. “Heterogeneous Stream Processing and Crowdsourcing for Urban Traffic Management.” In: *Proceedings of the International Conference on Extending Database Technology*. 2014, pp. 712–723 (cit. on p. 66).

[24] Marcos Dias de Assuncao, Alexandre da Silva Veith, and Rajkumar Buyya. “Distributed data stream processing and edge computing: A survey on resource elasticity and future directions.” In: *Journal of Network and Computer Applications* 103 (2018), pp. 1–17 (cit. on pp. 10, 15).

[25] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. “End-to-end differentiable physics for learning and control.” In: vol. 31. 2018 (cit. on pp. 4, 22).

[26] Joseph Azar, Abdallah Makhoul, Mahmoud Barhamgi, and Raphaël Couturier. “An energy efficient IoT data compression approach for edge machine learning.” In: *Future Generation Computer Systems* 96 (2019), pp. 168–175 (cit. on p. 135).

[27] Fuad Bajaber, Sherif Sakr, Omar Batarfi, Abdulrahman Altalhi, and Ahmed Barnawi. “Benchmarking big data systems: A survey.” In: *Computer Communications* 149 (2020), pp. 241–251 (cit. on pp. 15, 16).

[28] Stefano Belloni, Daniel Ritter, Marco Schröder, and Nils Rörup. “Deepbench: Benchmarking JSON document stores.” In: *Proceedings of the workshop on 9th International Workshop of Testing Database Systems*. 2022, pp. 1–9 (cit. on p. 17).

[29] Yael Ben-Haim and Elad Tom-Tov. “A Streaming Parallel Decision Tree Algorithm.” In: *Journal of Machine Learning Research* 11.2 (2010) (cit. on p. 21).

[30] Lars Bergstrom and John Reppy. “Nested Data-Parallelism on the Gpu.” In: *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming*. 2012, pp. 247–258 (cit. on p. 36).

[31] Sukanya Bhowmik, Muhammad Adnan Tariq, Boris Koldehofe, Frank Dürr, Thomas Kohler, and Kurt Rothermel. “High Performance Publish/Subscribe Middleware in Software-Defined Networks.” In: *IEEE/ACM Transactions on Networking* 25.3 (2017), pp. 1501–1516 (cit. on pp. 10, 133).

[32] Alain Biem, Eric Bouillet, Hanhua Feng, Anand Ranganathan, Anton Riabov, Olivier Verscheure, Haris Koutsopoulos, and Carlos Moran. “IBM infosphere streams for scalable, real-time, intelligent transportation services.” In: *Proceedings of the ACM SIGMOD International Conference on Management of data*. 2010, pp. 1093–1104 (cit. on pp. 57, 61).

[33] Netflix Technology Blog. *Keystone Real-time Stream Processing Platform*. `https://t.ly/61XZJ`. [Online; accessed 25-06-2024]. 2018 (cit. on pp. 1, 2, 11, 12, 28, 43, 45, 46, 71).

[34] Christoph Boden, Tilmann Rabl, Sebastian Schelter, and Volker Markl. “Benchmarking distributed data processing systems for machine learning workloads.” In: *Proceedings of the Performance Evaluation and Benchmarking for the Era of Artificial Intelligence: 10th TPC Technology Conference, TPCTC 2018*. 2019, pp. 42–57 (cit. on pp. 16, 17).

[35] Christoph Boden, Andrea Spina, Tilmann Rabl, and Volker Markl. “Benchmarking data flow systems for scalable machine learning.” In: *Proceedings of the 4th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond*. 2017, pp. 1–10 (cit. on p. 16).

[36] Peter Bodik, Wei Hong, Carlos Guestrin, Sam Madden, Mark Paskin, Romain Thibaux, Joe Polastre, and Rob Szewczyk. *Intel Lab Data*. [Online; accessed 25-06-2024]. 2004. URL: `http://db.csail.mit.edu/labdata/labdata.html` (cit. on p. 112).

[37] Nils Boeschen and Carsten Binnig. "GaccO - A GPU-accelerated OLTP DBMS." In: *Proceeding of the International Conference on Management of Data*. 2022, pp. 1003–1016 (cit. on p. 36).

[38] Peter Boncz, Thomas Neumann, and Orri Erling. "TPC-H analyzed: Hidden messages and lessons learned from an influential benchmark." In: *Technology Conference on Performance Evaluation and Benchmarking*. 2013, pp. 61–76 (cit. on p. 17).

[39] Peter Boncz, Thomas Neumann, and Orri Erling. "TPC-H analyzed: Hidden messages and lessons learned from an influential benchmark." In: *Proceedings of the Technology Conference on Performance Evaluation and Benchmarking*. 2013, pp. 61–76 (cit. on pp. 57, 66).

[40] Andrew Bond, Doug Johnson, Greg Kopczynski, and H Reza Taheri. "Profiling the performance of virtualized databases with the TPCx-V benchmark." In: *Proceedings of the Performance Evaluation and Benchmarking: Traditional to Big Data to Internet of Things: 7th TPC Technology Conference, TPCTC 2015, Kohala Coast, HI, USA, August 31–September 4, 2015. Revised Selected Papers 7*. 2016, pp. 156–172 (cit. on p. 17).

[41] Boris Jan Bonfils and Philippe Bonnet. "Adaptive and Decentralized Operator Placement for In-Network Query Processing." In: *Telecommunication Systems* 26.2 (2004), pp. 389–409 (cit. on p. 10).

[42] Maycon Viana Bordin, Dalvan Griebler, Gabriele Mencagli, Cláudio FR Geyer, and Luiz Gustavo L Fernandes. "Dspbench: A suite of benchmark applications for distributed data stream processing systems." In: *IEEE Access* 8 (2020), pp. 222900–222917 (cit. on pp. 2, 15, 16, 37, 44, 45, 56, 57, 64, 78, 112).

[43] Bochra Boughzala, Christoph Gärtner, and Boris Koldehofe. "Window-based parallel operator execution with in-network computing." In: *Proceedings of the 16th ACM International Conference on Distributed and Event-Based Systems*. 2022, pp. 91–96 (cit. on pp. 10, 133).

[44] Bochra Boughzala and Boris Koldehofe. "In-Network Management of Parallel Data Streams over Programmable Data Planes." In: *Proceedings of the 23rd International Federation for Information Processing (IFIP) Networking Conference (IFIP NETWORKING 2024)*. IEEE, 2024 (cit. on p. 10).

[45] Leo Breiman. "Random forests." In: *Machine learning* 45 (2001), pp. 5–32 (cit. on pp. 21, 75, 102).

[46] 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: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. 2020, pp. 1877–1901 (cit. on pp. 74, 131).

[47] Alejandro Buchmann and Boris Koldehofe. "Complex Event Processing." In: *IT - Information Technology* 51.5 (2009), pp. 241–242 (cit. on pp. 9, 10).

[48] Paul Cao, Bhaskar Gowda, Seetha Lakshmi, Chinmayi Narasimhadevara, Patrick Nguyen, John Poelman, Meikel Poess, and Tilmann Rabl. "From bigbench to TPCx-BB: Standardization of a big data benchmark." In: *Proceedings of the Performance Evaluation and Benchmarking. Traditional-Big Data-Internet of Things: 8th TPC Technology Conference, TPCTC 2016*. 2017, pp. 24–44 (cit. on p. 17).

[49] Paris Carbone, Stephan Ewen, Gyula Fóra, Seif Haridi, Stefan Richter, and Kostas Tzoumas. "State Management in Apache Flink®: Consistent Stateful Distributed Stream Processing." In: *Proceedings of the VLDB Endowment* 10.12 (2017), pp. 1718–1729 (cit. on p. 71).

[50] Paris Carbone, Stephan Ewen, Gyula Fóra, Seif Haridi, Stefan Richter, and Kostas Tzoumas. "State management in Apache Flink®: consistent stateful distributed stream processing." In: *Proceedings of the VLDB Endowment* 10.12 (2017), pp. 1718–1729 (cit. on pp. 6, 35).

[51] Paris Carbone, Marios Fragkoulis, Vasiliki Kalavri, and Asterios Katsifodimos. "Beyond analytics: The evolution of stream processing systems." In: *Proceedings of the ACM SIGMOD international conference on Management of data*. 2020, pp. 2651–2658 (cit. on p. 10).

[52] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. "Apache flink: Stream and batch processing in a single engine." In: *The Bulletin of the Technical Committee on Data Engineering* 38.4 (2015), pp. 28–38 (cit. on pp. 6, 18, 31, 33, 35, 43, 47, 80, 96, 98, 100).

[53] Valeria Cardellini, Vincenzo Grassi, Francesco Lo Presti, and Matteo Nardelli. "Optimal operator replication and placement for distributed stream processing systems." In: *ACM SIGMETRICS Performance Evaluation Review* 44.4 (2017), pp. 11–22 (cit. on pp. 15, 18, 19, 78).

[54] Valeria Cardellini, Francesco Lo Presti, Matteo Nardelli, and Gabriele Russo Russo. "Decentralized Self-Adaptation for Elastic Data Stream Processing." In: *Future Generation Computer System* 87.C (2018), pp. 171–185 (cit. on pp. 4, 15, 19, 23, 72, 73, 134).

[55] Carmen Carrión. "Kubernetes scheduling: Taxonomy, ongoing issues and challenges." In: *ACM Computing Surveys* 55.7 (2022), pp. 1–37 (cit. on p. 98).

[56] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. "Design and Evaluation of a Wide-Area Event Notification Service." In: *ACM Transactions on Computer Systems* 19.3 (2001), pp. 332–383 (cit. on pp. 43, 44).

[57] Jared Casper and Kunle Olukotun. "Hardware Acceleration of Database Operations." In: *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2014, pp. 151–160 (cit. on p. 36).

[58] Guoqiang Jerry Chen, Janet L Wiener, Shridhar Iyer, Anshul Jaiswal, Ran Lei, Nikhil Simha, Wei Wang, Kevin Wilfong, Tim Williamson, and Serhat Yilmaz. "Realtime data processing at facebook." In: *Proceedings of the International Conference on Management of Data*. 2016, pp. 1087–1098 (cit. on p. 1).

[59] Jianguo Chen, Kenli Li, Zhuo Tang, Kashif Bilal, Shui Yu, Chuliang Weng, and Keqin Li. "A parallel random forest algorithm for big data in a spark cloud computing environment." In: *IEEE Transactions on Parallel and Distributed Systems* 28.4 (2016), pp. 919–933 (cit. on pp. 21, 95, 108).

[60] Ning Chen, Yu Chen, Yang You, Haibin Ling, Pengpeng Liang, and Roger Zimmermann. "Dynamic urban surveillance video stream processing using fog computing." In: *Proceedings of the IEEE second international conference on multimedia big data (BigMM)*. 2016, pp. 105–112 (cit. on p. 30).

[61] Shimin Chen, Anastasia Ailamaki, Manos Athanassoulis, Phillip B Gibbons, Ryan Johnson, Ippokratis Pandis, and Radu Stoica. "TPC-E vs. TPC-C: Characterizing the new TPC-E benchmark via an I/O comparison study." In: *ACM Sigmod Record* 39.3 (2011), pp. 5–10 (cit. on p. 17).

[62] Dazhao Cheng, Yuan Chen, Xiaobo Zhou, Daniel Gmach, and Dejan Milojicic. "Adaptive scheduling of parallel jobs in spark streaming." In: *Proceedings of the IEEE Conference on Computer Communications*. 2017, pp. 1–9 (cit. on p. 23).

[63] Dazhao Cheng, Xiaobo Zhou, Yu Wang, and Changjun Jiang. "Adaptive scheduling parallel jobs with dynamic batching in spark streaming." In: *IEEE Transactions on Parallel and Distributed Systems* 29.12 (2018), pp. 2672–2685 (cit. on p. 23).

[64] Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar, Thomas Graves, Mark Holderbaugh, Zhuo Liu, Kyle Nusbaum, Kishorkumar Patil, Boyang Jerry Peng, et al. "Benchmarking streaming computation engines: Storm, flink and spark streaming." In: *Proceedings of the IEEE international parallel and distributed processing symposium workshops (IPDPSW)*. 2016, pp. 1789–1792 (cit. on pp. 2, 16, 44, 45, 56, 58).

[65] Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar, Tom Graves, Mark Holderbaugh, Zhuo Liu, Kyle Nusbaum, Kishorkumar Patil, Boyang Peng, et al. "Benchmarking streaming computation engines at yahoo." In: *Technical Report* (2015) (cit. on p. 16).

[66] Gianpaolo Cugola and Alessandro Margara. "Processing flows of information: From data stream to complex event processing." In: *ACM Computing Surveys (CSUR)* 44.3 (2012), pp. 1–62 (cit. on pp. 10, 12, 13, 15, 18).

[67] Wenyun Dai, Longfei Qiu, Ana Wu, and Meikang Qiu. "Cloud Infrastructure Resource Allocation for Big Data Applications." In: *IEEE Transactions on Big Data* 4.3 (2018), pp. 313–324 (cit. on pp. 4, 15, 19).

[68] Miyuru Dayarathna and Toyotaro Suzumura. "Automatic optimization of stream programs via source program operator graph transformations." In: *Distributed and Parallel Databases* 31 (2013), pp. 543–599 (cit. on p. 61).

[69] Daniele De Sensi, Tiziano De Matteis, and Marco Danelutto. "Simplifying self-adaptive and power-aware computing with Nornir." In: *Future Generation Computer Systems* 87 (2018), pp. 136–151 (cit. on p. 20).

[70] ACM DEBS. *DEBS 2014 Grand Challenge: Smart homes*. `https://debs.org/grand-challenges/2014/`. [Online; accessed 25-04-2024]. 2016 (cit. on pp. 57, 65).

[71] DEBS.org. *DEBS Grand Challenge*. [Online; accessed 25-06-2024]. 2023 (cit. on p. 51).

[72] Yahya Al-Dhuraibi, Fawaz Paraiso, Nabil Djarallah, and Philippe Merle. "Elasticity in cloud computing: state of the art and research challenges." In: *IEEE Transactions on services computing* 11.2 (2017), pp. 430–447 (cit. on pp. 18, 23).

[73] DOMO. *Data Never Sleeps 11.0*. `https://www.domo.com/learn/infographic/data-never-sleeps-11`. [Online; accessed 25-06-2024]. 2022 (cit. on p. 29).

[74] Lixin Duan, Dong Xu, and Ivor Tsang. "Learning with augmented features for heterogeneous domain adaptation." In: *arXiv preprint arXiv:1206.4660* (2012) (cit. on p. 25).

[75] Saurabh Dubey. *Real Time Sentiment Analysis*. `https://github.com/voltas/real-time-sentiment-analytic`. [Online; accessed 25-04-2024]. 2015 (cit. on pp. 57, 64).

[76] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. "The Design and Operation of CloudLab." In: *Proceedings of the USENIX Conference on Annual Technical Conference*. 2019, pp. 1–14 (cit. on pp. 4, 5, 31, 46, 52, 97, 98, 100, 131).

[77] Rahul Dwarakanath, Boris Koldehofe, Yashas Bharadwaj, The An Binh Nguyen, David Eyers, and Ralf Steinmetz. "TrustCEP: Adopting a trust-based approach for distributed complex event processing." In: *Proceedings of the 18th IEEE international conference on mobile data management (MDM)*. IEEE. 2017, pp. 30–39 (cit. on p. 9).

[78] Rahul Chini Dwarakanath, Boris Koldehofe, and Ralf Steinmetz. "Operator Migration for Distributed Complex Event Processing in Device-to-Device Based Networks." In: *M4IoT@ Middleware*. 2016, pp. 13–18 (cit. on p. 15).

[79] Wolfgang Effelsberg, Ralf Steinmetz, and Thorsten Strufe. *Benchmarking Peer-to-Peer Systems*. Springer, 2013 (cit. on pp. 15, 43, 44).

[80] Zhiwei Fan, Rathijit Sen, Paraschos Koutris, and Aws Albarghouthi. "Automated Tuning of Query Degree of Parallelism via Machine Learning." In: *Proceedings of the 3rd International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. 2020 (cit. on p. 23).

[81] Paulo Figueiras, Zala Herga, Guilherme Guerreiro, António Rosa, Ruben Costa, and Ricardo Jardim-Gonçalves. "Real-time monitoring of road traffic using data stream mining." In: *Proceedings of the IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. 2018, pp. 1–8 (cit. on p. 1).

[82] Data Flair. *Apache Flink Use Cases – Real life case studies of Apache Flink*. [Online; accessed 25-06-2024]. 2016 (cit. on pp. 20, 46, 96).

[83] Avrilia Floratou, Ashvin Agrawal, Bill Graham, Sriram Rao, and Karthik Ramasamy. "Dhalion: Self-Regulating Stream Processing in Heron." In: *Proceedings of VLDB Endowment* 10.12 (2017), pp. 1825–1836 (cit. on pp. 6, 19, 72, 75, 77, 89, 93, 95, 102, 124, 125, 132).

[84] Daniele Foroni, Cristian Axenie, Stefano Bortoli, Mohamad Al Hajj Hassan, Ralph Acker, Radu Tudoran, Goetz Brasche, and Yannis Velegrakis. "Moira: A goal-oriented incremental machine learning approach to dynamic resource cost estimation in distributed stream processing systems." In: *Proceedings of the international workshop on real-time business intelligence and analytics*. 2018, pp. 1–10 (cit. on p. 23).

[85] Sebastian Frischbier, Mario Paic, Alexander Echler, and Christian Roth. "Managing the Complexity of Processing Financial Data at Scale - An Experience Report." In: *Proceedings of the Complex Systems Design and Management*. 2019, pp. 14–26 (cit. on pp. 9, 11, 30).

[86] Sebastian Frischbier, Mario Paic, Alexander Echler, and Christian Roth. "Managing the complexity of processing financial data at scale-an experience report." In: *Complex Systems Design & Management: Proceedings of the Tenth International Conference on Complex Systems Design & Management, CSD&M Paris 2019*. 2020, pp. 14–26 (cit. on p. 20).

[87] Alexander Frömmgen, Robert Rehner, Max Lehn, and Alejandro Buchmann. "Fossa: Learning ECA rules for adaptive distributed systems." In: *Proceedings of the IEEE International Conference on Autonomic Computing*. 2015, pp. 207–210 (cit. on p. 23).

[88] Alexander Frömmgen, Björn Richerzhagen, Julius Rückert, David Hausheer, Ralf Steinmetz, and Alejandro Buchmann. "Towards the description and execution of transitions in networked systems." In: *Intelligent Mechanisms for Network Configuration and Security: 9th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2015, Ghent, Belgium, June 22-25, 2015. Proceedings 9*. 2015, pp. 17–29 (cit. on p. 2).

[89] Alexander Frömmgen, Denny Stohr, Boris Koldehofe, and Amr Rizk. "Don't repeat yourself: seamless execution and analysis of extensive network experiments." In: *Proceedings of the 14th ACM International Conference on emerging Networking Experiments and Technologies (CONEXT)*. 2018, pp. 20–26 (cit. on p. 15).

[90] Archana Ganapathi, Harumi Kuno, Umeshwar Dayal, Janet L. Wiener, Armando Fox, Michael Jordan, and David Patterson. "Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning." In: *Proceedings of the IEEE 25th International Conference on Data Engineering*. 2009, pp. 592–603 (cit. on pp. 75, 93, 95, 102, 108, 132).

[91] Adriano Marques Garcia, Dalvan Griebler, Claudio Schepke, and Luiz Gustavo Fernandes. "SPBench: a framework for creating benchmarks of stream processing applications." In: *Springer Computing* 105.5 (2023), pp. 1077–1099 (cit. on pp. 2, 16, 44, 45, 56, 57).

[92] Christoph Gärtner, Amr Rizk, Boris Koldehofe, René Guillaume, Ralf Kundel, and Ralf Steinmetz. "On the Incremental Reconfiguration of Time-sensitive Networks at Runtime." In: *Proceedings of the IFIP Networking Conference*. 2022, pp. 1–9 (cit. on p. 36).

[93] Christoph Gärtner, Amr Rizk, Boris Koldehofe, René Guillaume, Ralf Kundel, and Ralf Steinmetz. "Demo: Flexibility-aware Network Management of Time-Sensitive Flows." In: *Proceedings of the ACM SIGCOMM Conference*. 2023, pp. 1176–1178 (cit. on p. 36).

[94] Christoph Gärtner, Amr Rizk, Boris Koldehofe, René Guillaume, Ralf Kundel, and Ralf Steinmetz. "Fast incremental reconfiguration of dynamic time-sensitive networks at runtime." In: *Computer Networks* 224 (Apr. 2023) (cit. on pp. 10, 36).

[95] Christoph Gärtner, Amr Rizk, Boris Koldehofe, Rhaban Hark, René Guillaume, and Ralf Steinmetz. "Leveraging Flexibility of Time-Sensitive Networks for dynamic Reconfigurability." In: *Proceedings of the IFIP Networking Conference*. 2021 (cit. on p. 23).

[96] Bugra Gedik, Henrique Andrade, Kun-Lung Wu, Philip S Yu, and Myungcheol Doo. "SPADE: The System S declarative stream processing engine." In: *Proceedings of the ACM SIGMOD international conference on Management of data*. 2008, pp. 1123–1134 (cit. on p. 61).

[97] Sandra Geisler and Christoph Quix. "Evaluation of real-time traffic applications based on data stream mining." In: *Data Mining for Geoinformatics: Methods and Applications* (2014), pp. 83–103 (cit. on p. 66).

[98] GENI. *Geni network infrastructure.* `https://www.geni.net/`. [Online; accessed 25-06-2024]. 2024 (cit. on p. 98).

[99] Mehdi Gheisari, Guojun Wang, and Md Zakirul Alam Bhuiyan. “A survey on deep learning in big data.” In: *Proceedings of the IEEE international conference on computational science and engineering (CSE) and IEEE international conference on embedded and ubiquitous computing (EUC).* 2017, pp. 173–180 (cit. on pp. 19, 73).

[100] Pegah Golchin, Chengbo Zhou, Pratyush Agnihotri, Mehrdad Hajizadeh, Ralf Kundel, and Ralf Steinmetz. “CML-IDS: Enhancing Intrusion Detection in SDN Through Collaborative Machine Learning.” In: *Proceedings of the 19th International Conference on Network and Service Management (CNSM).* Best Paper Award. 2023, pp. 1–9 (cit. on p. 133).

[101] Dalvan Griebler, Adriano Vogel, Daniele De Sensi, Marco Danelutto, and Luiz G Fernandes. “Simplifying and implementing service level objectives for stream parallelism.” In: *The Journal of Supercomputing* 76.6 (2020), pp. 4603–4628 (cit. on p. 20).

[102] Jamie Grier. “Extending the yahoo! streaming benchmark.” In: *URL http://data-artisans. com/extending-the-yahoo-streamingbenchmark* (2016) (cit. on p. 16).

[103] Vincenzo Gulisano, Ricardo Jimenez-Peris, Marta Patino-Martinez, Claudio Soriente, and Patrick Valduriez. “Streamcloud: An elastic and scalable data streaming system.” In: *IEEE Transactions on Parallel and Distributed Systems* 23.12 (2012), pp. 2351–2365 (cit. on p. 19).

[104] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. “Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints.” In: *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services.* 2016, pp. 123–136 (cit. on p. 21).

[105] Maayan Harel and Shie Mannor. “Learning from multiple outlooks.” In: *arXiv preprint arXiv:1005.0027* (2010) (cit. on p. 25).

[106] Roman Heinrich, Carsten Binnig, Harald Kornmayer, and Manisha Luthra. “COSTREAM: Learned Cost Models for Operator Placement in Edge-Cloud Environments.” In: *Proceedings of the IEEE 40th International Conference on Data Engineering (ICDE).* 2024, pp. 1–16 (cit. on pp. 22, 23, 45, 46, 108, 109, 131).

[107] Roman Heinrich, Manisha Luthra, Harald Kornmayer, and Carsten Binnig. “Zero-shot cost models for distributed stream processing.” In: *Proceedings of the 16th ACM International Conference on Distributed and Event-Based Systems.* 2022, pp. 85–90 (cit. on pp. 16, 19, 22, 23, 25, 44, 45).

[108] Thomas Heinze, Zbigniew Jerzak, Gregor Hackenbroich, and Christof Fetzer. “Latency-Aware Elastic Scaling for Distributed Data Stream Processing Systems.” In: *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems.* 2014, pp. 13–22 (cit. on pp. 19, 23, 72, 73).

[109] Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. “Towards the systematic reporting of the energy and carbon footprints of machine learning.” In: *Journal of Machine Learning Research* 21.248 (2020), pp. 1–43 (cit. on p. 134).

[110] Guenter Hesse and Martin Lorenz. “Conceptual Survey on Data Stream Processing Systems.” In: *Proceedings of the IEEE 21st International Conference on Parallel and Distributed Systems.* IEEE Computer Society, 2015, pp. 797–802 (cit. on p. 16).

[111] Guenter Hesse, Christoph Matthies, Michael Perscheid, Matthias Uflacker, and Hasso Plattner. "Espbench: the enterprise stream processing benchmark." In: *Proceedings of the ACM/SPEC International Conference on Performance Engineering.* 2021, pp. 201–212 (cit. on pp. 2, 15, 37, 45, 56, 57).

[112] Nicolas Hidalgo, Daniel Wladdimiro, and Erika Rosas. "Self-Adaptive Processing Graph with Operator Fission for Elastic Stream Processing." In: *Journal of Systems and Software* 127.C (2017), pp. 205–216 (cit. on pp. 4, 15, 19, 23).

[113] Benjamin Hilprecht and Carsten Binnig. "One Model to Rule them All: Towards Zero-Shot Learning for Databases." In: *Processing of the 12th Conference on Innovative Data Systems Research.* 2022 (cit. on pp. 22, 76).

[114] Benjamin Hilprecht and Carsten Binnig. "Zero-Shot Cost Models for out-of-the-Box Learned Cost Prediction." In: *Proceeding of the VLDB Endowment* 15.11 (2022), pp. 2361–2374 (cit. on pp. 22, 25, 73, 76, 77, 81, 87, 102).

[115] Benjamin Hilprecht, Carsten Binnig, Tiemo Bang, Muhammad El-Hindi, Benjamin Hättasch, Aditya Khanna, Robin Rehrmann, Uwe Röhm, Andreas Schmidt, Lasse Thostrup, et al. "DBMS Fitting: Why should we learn what we already know?" In: *Proceedings of the 10th Conference on Innovative Data Systems Research.* 2020 (cit. on p. 22).

[116] 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: *Proceedings of the 10th Conference on Innovative Data Systems Research.* 2020 (cit. on p. 21).

[117] Benjamin Hilprecht, Carsten Binnig, and Uwe Röhm. "Towards Learning a Partitioning Advisor with Deep Reinforcement Learning." In: *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management.* 2019 (cit. on p. 23).

[118] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. "DeepDB: Learn from Data, Not from Queries!" In: *Proceeding of the VLDB Endowment* 13.7 (2020), pp. 992–1005 (cit. on p. 21).

[119] Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik, and Robert Grimm. "A catalog of stream processing optimizations." In: *ACM Computing Surveys (CSUR)* 46.4 (2014), pp. 1–34 (cit. on p. 34).

[120] P Hosseinzadeh Talaee. "Multilayer perceptron with different training algorithms for streamflow forecasting." In: *Neural Computing and Applications* 24 (2014), pp. 695–703 (cit. on pp. 21, 95, 108).

[121] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis." In: *Proceedings of the IEEE 26th International Conference on Data Engineering Workshops ICDEW.* 2010, pp. 41–51 (cit. on pp. 16, 44, 45).

[122] Stratos Idreos, Kostas Zoumpatianos, Brian Hentschel, Michael S Kester, and Demi Guo. "The data calculator: Data structure design and cost synthesis from first principles and learned cost models." In: *Proceedings of the International Conference on Management of Data.* 2018, pp. 535–550 (cit. on p. 18).

[123] Nina Ihde, Paula Marten, Ahmed Eleliemy, Gabrielle Poerwawinata, Pedro Silva, Ilin Tolovski, Florina M Ciorba, and Tilmann Rabl. “A survey of big data, high performance computing, and machine learning benchmarks.” In: *Proceedings of the Performance Evaluation and Benchmarking: 13th TPC Technology Conference, TPCTC 2021, Copenhagen, Denmark, August 20, 2021, Revised Selected Papers 13*. 2022, pp. 98–118 (cit. on pp. 16, 57).

[124] Muhammad Hussain Iqbal, Tariq Rahim Soomro, et al. “Big data analysis: Apache storm perspective.” In: *International journal of computer trends and technology* 19.1 (2015), pp. 9–14 (cit. on pp. 18, 29, 31, 97).

[125] Haruna Isah, Tariq Abughofa, Sazia Mahfuz, Dharmitha Ajerla, Farhana Zulkernine, and Shahzad Khan. “A survey of distributed data stream processing frameworks.” In: *IEEE Access* 7 (2019), pp. 154300–154316 (cit. on pp. 15, 16).

[126] Matthias Jasny, Lasse Thostrup, Tobias Ziegler, and Carsten Binnig. “P4DB - The Case for In-Network OLTP.” In: *Proceeding of the International Conference on Management of Data*. 2022, pp. 1375–1389 (cit. on p. 36).

[127] Xinrui Jiang, Nannan Wang, Jingwei Xin, Xiaobo Xia, Xi Yang, and Xinbo Gao. “Learning lightweight super-resolution networks with weight pruning.” In: *Neural Networks* 144 (2021), pp. 21–32 (cit. on p. 135).

[128] Yx Jiang. *Real Time Anomaly Detection Framework*. `https://github.com/yxjiang/stream-outlier`. [Online; accessed 25-04-2024]. 2013 (cit. on pp. 57, 64).

[129] Gideon Juve, Ewa Deelman, Karan Vahi, Gaurang Mehta, Bruce Berriman, Benjamin P Berman, and Phil Maechling. “Scientific workflow applications on Amazon EC2.” In: *Proceedings of the 5th IEEE international conference on e-science workshops*. 2009, pp. 59–66 (cit. on p. 31).

[130] Sabihe Kabirzadeh, Dadmehr Rahbari, and Mohsen Nickray. “A hyper heuristic algorithm for scheduling of fog networks.” In: *Proceedings of the 21st Conference of Open Innovations Association (FRUCT)*. 2017, pp. 148–155 (cit. on pp. 15, 18, 19).

[131] Vasiliki Kalavri, John Liagouris, Moritz Hoffmann, Desislava Dimitrova, Matthew Forshaw, and Timothy Roscoe. “Three steps is all you need: fast, accurate, automatic scaling decisions for distributed streaming dataflows.” In: *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation*. 2018, pp. 783–798 (cit. on pp. 4, 6, 19, 56, 72, 73, 75, 77, 89).

[132] Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen, and Volker Markl. “Benchmarking Distributed Stream Data Processing Systems.” In: *Proceedings of the IEEE 34th international conference on data engineering (ICDE)*. 2018, pp. 1507–1518 (cit. on pp. 2, 16, 44, 56).

[133] Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen, and Volker Markl. “Benchmarking Distributed Stream Data Processing Systems.” In: *Proceeding of the IEEE 34th International Conference on Data Engineering*. 2018, pp. 1507–1518 (cit. on pp. 37, 44, 78).

[134] Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen, and Volker Markl. “Benchmarking distributed stream data processing systems.” In: *Proceedings of the IEEE 34th international conference on data engineering (ICDE)*. 2018, pp. 1507–1518 (cit. on pp. 16, 57, 62).

[135] Junaid Ahmed Khan, Cedric Westphal, and Yacine Ghamri-Doudane. “Offloading content with self-organizing mobile fogs.” In: *Proceedings of the 29th International Teletraffic Congress (ITC)*. 2017, pp. 223–231 (cit. on pp. 15, 19).

[136] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. “Learned cardinalities: Estimating correlated joins with deep learning.” In: *arXiv preprint arXiv:1809.00677* (2018) (cit. on p. 21).

[137] Kristian Kloeckl, Oliver Senn, and Carlo Ratti. “Enabling the real-time city: LIVE Singapore!” In: *Street Computing*. 2016, pp. 86–109 (cit. on p. 30).

[138] Boris Koldehofe, Ruben Mayer, Umakishore Ramachandran, Kurt Rothermel, and Marco Völz. “Rollback-recovery without checkpoints in distributed event processing systems.” In: *Proceedings of the 7th ACM international conference on Distributed event-based systems (DEBS)*. 2013, pp. 27–38 (cit. on p. 10).

[139] Alexandros Koliousis, Matthias Weidlich, Raul Castro Fernandez, Alexander L Wolf, Paolo Costa, and Peter Pietzuch. “Saber: Window-based hybrid stream processing for heterogeneous architectures.” In: *Proceedings of the International Conference on Management of Data*. 2016, pp. 555–569 (cit. on pp. 1, 2, 29, 36, 133).

[140] Roland Kotto Kombi, Nicolas Lumineau, and Philippe Lamarre. “A Preventive Auto-Parallelization Approach for Elastic Stream Processing.” In: *Proceedings of the IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2017, pp. 1532–1542 (cit. on pp. 4, 15, 19, 23, 95).

[141] Mikołaj Komisarek, Michał Choraś, Rafał Kozik, and Marek Pawlicki. “Real-time stream processing tool for detecting suspicious network patterns using machine learning.” In: *Proceedings of the 15th International Conference on Availability, Reliability and Security*. 2020, pp. 1–7 (cit. on p. 21).

[142] Jay Kreps, Neha Narkhede, Jun Rao, et al. “Kafka: A distributed messaging system for log processing.” In: *Proceedings of the NetDB*. 2011, pp. 1–7 (cit. on p. 63).

[143] Brian Kulis, Kate Saenko, and Trevor Darrell. “What you saw is not what you get: Domain adaptation using asymmetric kernel transforms.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2011, pp. 1785–1792 (cit. on p. 25).

[144] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M Patel, Karthik Ramasamy, and Siddarth Taneja. “Twitter heron: Stream processing at scale.” In: *Proceedings of the ACM SIGMOD international conference on Management of data*. 2015, pp. 239–250 (cit. on pp. 18, 33, 47, 57, 67, 125, 133).

[145] Ralf Kundel, Christoph Gärtner, Manisha Luthra, Sukanya Bhowmik, and Boris Koldehofe. “Flexible Content-based Publish/Subscribe over Programmable Data Planes.” In: *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*. 2020, pp. 1–5 (cit. on pp. 10, 18).

[146] Ralf Kundel, Leonhard Nobach, Jeremias Blendin, Hans-Joerg Kolbe, Georg Schyguda, Vladimir Gurevich, Boris Koldehofe, and Ralf Steinmetz. “P4-BNG: Central Office Network Functions on Programmable Packet Pipelines.” In: *Proceedings of the 15th International Conference on Network and Service Management (CNSM)*. 2019, pp. 1–9 (cit. on p. 133).

[147] Ralf Kundel, Fridolin Siegmund, Jeremias Blendin, Amr Rizk, and Boris Koldehofe. “P4STA: High Performance Packet Timestamping with Programmable Packet Processors.” In: *Proceedings of the IEEE/IFIP Network Operations and Management Symposium*. 2020, pp. 1–9 (cit. on pp. 10, 133).

[148] Ralf Kundel, Fridolin Siegmund, Rhaban Hark, Amr Rizk, and Boris Koldehofe. "Network Testing Utilizing Programmable Network Hardware." In: *IEEE Communications Magazine* 60.2 (2022), pp. 12–17 (cit. on p. 36).

[149] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. "How Good Are Query Optimizers, Really?" In: *Proceeding of the VLDB Endowment* 9.3 (2015), pp. 204–215 (cit. on p. 101).

[150] Viktor Leis, Bernhard Radke, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. "Query optimization through the looking glass, and what we found running the Join Order Benchmark." In: *VLDB Journal* 27.5 (2018), pp. 643–668 (cit. on p. 23).

[151] Alberto Lerner, Matthias Jasny, Theo Jepsen, Carsten Binnig, and Philippe Cudré-Mauroux. "DBMS annihilator: a high-performance database workload generator in action." In: *Proceeding of the VLDB Endowment* 15.12 (2022), pp. 3682–3685 (cit. on pp. 16, 43).

[152] Changlong Li, Hang Zhuang, Qingfeng Wang, and Xuehai Zhou. "SSLB: Self-Similarity-Based Load Balancing for Large-Scale Fog Computing." In: *Arabian Journal for Science & Engineering (Springer Science & Business Media BV)* 43.12 (2018) (cit. on pp. 15, 19).

[153] Feifei Li. "Cloud-Native Database Systems at Alibaba: Opportunities and Challenges." In: *Proceedings of VLDB Endowment* 12.12 (2019), pp. 2263–2272 (cit. on pp. 20, 71).

[154] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. "CleanML: A study for evaluating the impact of data cleaning on ml classification tasks." In: *Proceedings of the IEEE 37th International Conference on Data Engineering (ICDE)*. 2021, pp. 13–24 (cit. on p. 17).

[155] Tzu-Mao Li, Michaël Gharbi, Andrew Adams, Frédo Durand, and Jonathan Ragan-Kelley. "Differentiable programming for image processing and deep learning in Halide." In: *ACM Transactions on Graphics (ToG)* 37.4 (2018), pp. 1–13 (cit. on pp. 4, 22).

[156] Geotools Library. *GeoTools*. `https://www.osgeo.org/projects/geotools/`. [Online; accessed 25-04-2024]. 2020 (cit. on pp. 57, 66).

[157] Liqing Liu, Zheng Chang, and Xijuan Guo. "Socially aware dynamic computation offloading scheme for fog computing system with energy harvesting devices." In: *IEEE Internet of Things Journal* 5.3 (2018), pp. 1869–1879 (cit. on p. 19).

[158] Xunyun Liu and Rajkumar Buyya. "Performance-oriented deployment of streaming applications on cloud." In: *IEEE Transactions on Big Data* 5.1 (2017), pp. 46–59 (cit. on p. 15).

[159] Xunyun Liu and Rajkumar Buyya. "Resource management and scheduling in distributed stream processing systems: a taxonomy, review, and future directions." In: *ACM Computing Surveys* 53.3 (2020), pp. 1–41 (cit. on pp. 12, 13, 15, 16).

[160] Giorgia Lodi, Leonardo Aniello, Giuseppe A Di Luna, and Roberto Baldoni. "An event-based platform for collaborative threats detection and monitoring." In: *Information Systems* 39 (2014), pp. 175–195 (cit. on p. 1).

[161] Haifeng Lu, Chunhua Gu, Fei Luo, Weichao Ding, and Xinping Liu. "Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning." In: *Future Generation Computer Systems* 102 (2020), pp. 847–861 (cit. on p. 135).

[162] Ruirui Lu, Gang Wu, Bin Xie, and Jingtong Hu. “Stream bench: Towards benchmarking modern distributed stream computing frameworks.” In: *Proceedings of the IEEE/ACM 7th International Conference on Utility and Cloud Computing*. 2014, pp. 69–78 (cit. on pp. 44, 45, 57, 61, 64).

[163] Manisha Luthra and Boris Koldehofe. “ProgCEP: A Programming Model for Complex Event Processing over Fog Infrastructure.” In: *Proceedings of the 2nd International Workshop on Distributed Fog Services Design (DFSD@Middleware)*. 2019, pp. 7–12 (cit. on p. 18).

[164] Manisha Luthra, Boris Koldehofe, Niels Danger, Pascal Weisenburger, Guido Salvaneschi, and Ioannis Stavrakakis. “TCEP: Transitions in Operator Placement to Adapt to Dynamic Network Environments.” In: *Journal of Computer and Systems Sciences (JCSS), Special Issue on Algorithmic Theory of Dynamic Networks and its Applications (accepted for publication)* (2020), pp. 1–76. URL: `https://luthramanisha.github.io/TCEP/` (cit. on pp. 2, 9, 15, 37, 78).

[165] Lucy Ellen Lwakatare, Aiswarya Raj, Ivica Crnkovic, Jan Bosch, and Helena Holmström Olsson. “Large-scale machine learning systems in real-world industrial settings: A review of challenges and solutions.” In: *Information and software technology* 127 (2020), p. 106368 (cit. on p. 134).

[166] Tobias Mann. *Amazon goes nuclear, acquires Cumulus Data’s atomic datacenters for 650M*. `https://www.theregister.com/2024/03/04/amazon_acquires_cumulus_nuclear_datacenter/`. [Online; accessed 25-06-2024]. 2024 (cit. on p. 134).

[167] Tobias Mann. *CEO of UK’s National Grid warns of datacenters’ thirst for power*. `https://www.theregister.com/2024/03/27/ceo_of_uks_national_grid/`. [Online; accessed 25-06-2024]. 2024 (cit. on p. 134).

[168] Samuel Marchal, Jérôme François, Radu State, and Thomas Engel. “PhishStorm: Detecting phishing with streaming analytics.” In: *IEEE Transactions on Network and Service Management* 11.4 (2014), pp. 458–471 (cit. on p. 30).

[169] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. “Neo: a learned query optimizer.” In: *Proceedings of the VLDB Endowment* 12.11 (2019), pp. 1705–1718 (cit. on p. 21).

[170] Alessandro Margara, Gianpaolo Cugola, Nicolò Felicioni, and Stefano Cilloni. “A model and survey of distributed data-intensive systems.” In: *ACM Computing Surveys* 56.1 (2023), pp. 1–69 (cit. on p. 18).

[171] Michael Mathioudakis and Nick Koudas. “Twittermonitor: trend detection over the twitter stream.” In: *Proceedings of the ACM SIGMOD International Conference on Management of data*. 2010, pp. 1155–1158 (cit. on pp. 9, 57, 67).

[172] Ruben Mayer and Hans-Arno Jacobsen. “Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools.” In: *ACM Computing Surveys (CSUR)* 53.1 (2020), pp. 1–37 (cit. on pp. 18, 21).

[173] Ruben Mayer, Boris Koldehofe, and Kurt Rothermel. “Meeting predictable buffer limits in the parallel execution of event processing operators.” In: *Proceedings of the IEEE International Conference on Big Data (BigData)*. 2014, pp. 402–411 (cit. on pp. 10, 15).

[174] Ruben Mayer, Boris Koldehofe, and Kurt Rothermel. “Predictable low-latency event detection with parallel complex event processing.” In: *IEEE Internet of Things Journal* 2.4 (2015), pp. 274–286 (cit. on pp. 4, 19, 72).

[175] Douglas C Montgomery, Elizabeth A Peck, and G Geoffrey Vining. *Introduction to linear regression analysis.* John Wiley & Sons, 2021 (cit. on p. 20).

[176] Rene Mueller, Jens Teubner, and Gustavo Alonso. "Data Processing on FPGAs." In: 2.1 (2009), pp. 910–921 (cit. on p. 36).

[177] MN Mundada, Sunil Kumar, and AV Shekdar. "E-waste: a new challenge for waste management in India." In: *International journal of environmental studies* 61.3 (2004), pp. 265–279 (cit. on p. 30).

[178] Raghunath Othayoth Nambiar and Meikel Poess. "The Making of TPC-DS." In: *Proceedings of the 32nd International Conference on Very Large Data Bases.* 2006, pp. 1049–1058 (cit. on p. 17).

[179] Hamid Nasiri, Saeed Nasehi, and Maziar Goudarzi. "Evaluation of distributed stream processing frameworks for IoT applications in Smart Cities." In: *Journal of Big Data* 6.1 (2019), p. 52 (cit. on p. 30).

[180] Vladimir Nasteski. "An overview of the supervised machine learning methods." In: *Horizons. b* 4.51-62 (2017), p. 56 (cit. on pp. 18, 23).

[181] Leonardo Neumeyer, Bruce Robbins, Anish Nair, and Anand Kesari. "S4: Distributed stream computing platform." In: *Proceeding of the IEEE International Conference on Data Mining Workshops.* 2010, pp. 170–177 (cit. on pp. 57, 60, 112).

[182] Jeffrey Nickoloff and Stephen Kuenzli. *Docker in action.* Simon and Schuster, 2019 (cit. on p. 97).

[183] Matthew Nokleby, Haroon Raja, and Waheed U Bajwa. "Scaling-up distributed processing of data streams for machine learning." In: *Proceedings of the IEEE* 108.11 (2020), pp. 1984–2012 (cit. on p. 23).

[184] Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel, Kirak Hong, David Lillethun, and Umakishore Ramachandran. "MCEP: A Mobility-Aware Complex Event Processing System." In: *ACM Transactions on Internet Technology (TOIT)* 14.1 (2014), pp. 1–24 (cit. on p. 10).

[185] Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel, Kirak Hong, and Umakishore Ramachandran. "RECEP: Selection-Based Reuse for Distributed Complex Event Processing." In: *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems (DEBS).* 2014, pp. 59–70 (cit. on p. 10).

[186] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning." In: *IEEE Transactions on knowledge and data engineering* 22.10 (2009), pp. 1345–1359 (cit. on p. 24).

[187] Olga Papaemmanouil, Ugur Çetintemel, and John Jannotti. "Supporting Generic Cost Models for Wide-Area Stream Processing." In: *Proceedings of the IEEE 25th International Conference on Data Engineering.* 2009 (cit. on pp. 18, 19).

[188] David Patterson, Joseph Gonzalez, Urs Hölzle, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David R So, Maud Texier, and Jeff Dean. "The carbon footprint of machine learning training will plateau, then shrink." In: *Computer* 55.7 (2022), pp. 18–28 (cit. on p. 134).

[189] Meikel Poess and Chris Floyd. "New TPC benchmarks for decision support and web commerce." In: *ACM Sigmod Record* 29.4 (2000), pp. 64–71 (cit. on p. 17).

[190] Meikel Poess, Raghunath Nambiar, Karthik Kulkarni, Chinmayi Narasimhadevara, Tilmann Rabl, and Hans-Arno Jacobsen. "Analysis of tpcx-iot: The first industry standard benchmark for iot gateway systems." In: *Proceedings of the IEEE 34th International Conference on Data Engineering (ICDE).* 2018, pp. 1519–1530 (cit. on p. 17).

[191] Constantin Pohl, Philipp Götze, and Kai-Uwe Sattler. "A Cost Model for Data Stream Processing on Modern Hardware." In: *Proceedings of the International Workshop on Accelerating Analytics and Data Management Systems Using Modern Processor and Storage Architectures,(ADMS).* 2017 (cit. on pp. 18, 19).

[192] Junfei Qiu, Qihui Wu, Guoru Ding, Yuhua Xu, and Shuo Feng. "A survey of machine learning for big data processing." In: *EURASIP Journal on Advances in Signal Processing* 2016 (2016), pp. 1–16 (cit. on pp. 19, 73).

[193] Muhammad Mazhar Rathore, Hojae Son, Awais Ahmad, Anand Paul, and Gwanggil Jeon. "Real-Time Big Data Stream Processing Using GPU with Spark Over Hadoop Ecosystem." In: *International Journal of Parallel Programming* 46.3 (2018), pp. 630–646 (cit. on p. 36).

[194] Kamran Razavi, Manisha Luthra, Boris Koldehofe, Max Mühlhäuser, and Lin Wang. "FA2: Fast, Accurate Autoscaling for Serving Deep Learning Inference with SLA Guarantees." In: *Proceedings of the IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS).* 2022, pp. 146–159 (cit. on pp. 21, 23).

[195] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. "Mlperf inference benchmark." In: *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA).* 2020, pp. 446–459 (cit. on p. 17).

[196] Bjoern Richerzhagen, Boris Koldehofe, and Ralf Steinmetz. "Immense dynamism." In: *German Research* 37.2 (2015), pp. 24–27 (cit. on p. 2).

[197] Björn Richerzhagen, Dominik Stingl, Julius Ruckert, Ralf Steinmetz, et al. "Simonstrator: Simulation and prototyping platform for distributed mobile applications." In: *Proceedings of the 8th EAI International Conference on Simulation Tools and Techniques.* 2015, pp. 99–108 (cit. on pp. 15, 44).

[198] Henriette Röger and Ruben Mayer. "A Comprehensive Survey on Parallelization and Elasticity in Stream Processing." In: *ACM Computing Surveys* 52.2 (2019), p. 37 (cit. on pp. 9, 10, 12, 13, 15, 16, 18, 34).

[199] Sebastian Ruder, Matthew E Peters, Swabha Swayamdipta, and Thomas Wolf. "Transfer learning in natural language processing." In: *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: Tutorials.* 2019, pp. 15–18 (cit. on pp. 24, 25).

[200] Gabriele Russo Russo, Valeria Cardellini, and Francesco Lo Presti. "Reinforcement Learning Based Policies for Elastic Stream Processing on Heterogeneous Resources." In: *Proceedings of the 13th ACM international conference on distributed and event-based systems.* 2019, pp. 31–42 (cit. on pp. 15, 19, 23, 134).

[201] Gabriele Russo Russo, Valeria Cardellini, and Francesco Lo Presti. "Hierarchical Auto-scaling Policies for Data Stream Processing on Heterogeneous Resources." In: *ACM Transactions on Autonomous and Adaptive Systems* 18.4 (2023) (cit. on pp. 4, 15, 19, 23, 72, 73, 134).

[202] Mohammad Sadoghi, Rija Javed, Naif Tarafdar, Harsh Singh, Rohan Palaniappan, and Hans-Arno Jacobsen. "Multi-query Stream Processing on FPGAs." In: *Proceedings of the IEEE 28th International Conference on Data Engineering.* 2012, pp. 1229–1232 (cit. on p. 36).

[203] Mohammad Sadoghi, Martin Labrecque, Harsh Singh, Warren Shum, and Hans-Arno Jacobsen. "Efficient event processing through reconfigurable hardware for algorithmic trading." In: *Proceedings of the VLDB Endowment* 3.1-2 (2010), pp. 1525–1528 (cit. on p. 1).

[204] Salman Salloum, Ruslan Dautov, Xiaojun Chen, Patrick Xiaogang Peng, and Joshua Zhexue Huang. "Big data analytics on Apache Spark." In: *International Journal of Data Science and Analytics* 1.3 (2016), pp. 145–164 (cit. on p. 15).

[205] Salman Salloum, Ruslan Dautov, Xiaojun Chen, Patrick Xiaogang Peng, and Joshua Zhexue Huang. "Big data analytics on Apache Spark." In: *International Journal of Data Science and Analytics* 1 (2016), pp. 145–164 (cit. on pp. 18, 33).

[206] Suresh Sankaranarayanan, Joel JPC Rodrigues, Vijayan Sugumaran, Sergei Kozlov, et al. "Data flow and distributed deep neural network based low latency IoT-edge computation model for big data environment." In: *Engineering Applications of Artificial Intelligence* 94 (2020), p. 103785 (cit. on p. 21).

[207] Björn Schilling, Boris Koldehofe, and Kurt Rothermel. "Efficient and distributed rule placement in heavy constraint-driven event systems." In: *Proceedings of the 13th IEEE International Conference on High Performance Computing and Communications (HPCC-2011).* 2011, pp. 355–364 (cit. on pp. 10, 15, 18).

[208] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. "Green ai." In: *Communications of the ACM* 63.12 (2020), pp. 54–63 (cit. on p. 134).

[209] Mennan Selimi, Llorenç Cerdà Alabern, Felix Freitag, Luís Veiga, Arjuna Sathiaseelan, and Jon Crowcroft. "A lightweight service placement approach for community network micro-clouds." In: *Journal of Grid Computing* 17 (2019), pp. 169–189 (cit. on pp. 15, 18, 19).

[210] Bassem Sellami, Akram Hakiri, and Sadok Ben Yahia. "Deep Reinforcement Learning for energy-aware task offloading in join SDN-Blockchain 5G massive IoT edge network." In: *Future Generation Computer Systems* 137 (2022), pp. 363–379 (cit. on p. 135).

[211] Manali Shaha and Meenakshi Pawar. "Transfer learning for image classification." In: *Proceedings of the second international conference on electronics, communication and aerospace technology (ICECA).* 2018, pp. 656–660 (cit. on pp. 24, 25).

[212] C Imthyaz Sheriff, Twaseef Naqishbandi, and Angelina Geetha. "Healthcare informatics and analytics framework." In: *Proceedings of the International Conference on Computer Communication and Informatics (ICCCI).* 2015, pp. 1–6 (cit. on pp. 9, 29).

[213] Jungeun Shin, Diana Arroyo, Asser Tantawi, Chen Wang, Alaa Youssef, and Rakesh Nagi. "Cloud-Native Workflow Scheduling Using a Hybrid Priority Rule and Dynamic Task Parallelism." In: *Proceedings of the 13th Symposium on Cloud Computing.* 2022, pp. 72–77 (cit. on pp. 4, 19).

[214] Anshu Shukla, Shilpa Chaturvedi, and Yogesh Simmhan. "Riotbench: An iot benchmark for distributed stream processing systems." In: *Concurrency and Computation: Practice and Experience* 29.21 (2017), pp. 42–57 (cit. on pp. 16, 45, 58, 65).

[215] Yogesh Simmhan, Baohua Cao, Michail Giakkoupis, and Viktor K Prasanna. "Adaptive rate stream processing for smart grid applications on clouds." In: *Proceedings of the 2nd international workshop on Scientific cloud computing.* 2011, pp. 33–38 (cit. on pp. 57, 65).

[216] Davide Simoncelli, Maurizio Dusi, Francesco Gringoli, and Saverio Niccolini. "Scaling out the performance of service monitoring applications with BlockMon." In: *International Conference on Passive and Active Network Measurement*. 2013, pp. 253–255 (cit. on p. 67).

[217] Olena Skarlat, Matteo Nardelli, Stefan Schulte, Michael Borkowski, and Philipp Leitner. "Optimized IoT service placement in the fog." In: *Service Oriented Computing and Applications* 11 (2017), pp. 427–443 (cit. on pp. 15, 19, 73).

[218] Domenico Solazzo. *Storm Log Processing*. `https://github.com/domenicosolazzo/click-topology`. [Online; accessed 25-04-2024]. 2013 (cit. on pp. 57, 63).

[219] Yan-Yan Song and LU Ying. "Decision tree methods: applications for classification and prediction." In: *Shanghai archives of psychiatry* 27.2 (2015), p. 130 (cit. on p. 21).

[220] PayPal Editorial Staff. *The power of data: How PayPal leverages machine learning to tackle fraud*. `https://www.paypal.com/us/brc/article/paypal-machine-learning-stop-fraud`. [Online; accessed 25-04-2024]. 2021 (cit. on p. 30).

[221] Michael Stonebraker, Ugur Çetintemel, and Stan Zdonik. "The 8 Requirements of Real-Time Stream Processing." In: *SIGMOD Record* 34.4 (2005), pp. 42–47 (cit. on pp. 37, 78).

[222] Eliza Strickland. "Andrew Ng, AI Minimalist: The Machine-Learning Pioneer Says Small is the New Big." In: *IEEE Spectrum* 59.4 (2022), pp. 22–50 (cit. on pp. 74, 131).

[223] Ji Sun and Guoliang Li. "An end-to-end learning-based cost estimator." In: *Proceedings of the VLDB Endowment* 13.3 (2019), pp. 307–319 (cit. on p. 21).

[224] H Reza Taheri, Gary Little, Bhavik Desai, Andrew Bond, Doug Johnson, and Greg Kopczynski. "Characterizing the performance and resilience of HCI clusters with the TPCx-HCI benchmark." In: *Proceedings of the Performance Evaluation and Benchmarking for the Era of Artificial Intelligence: 10th TPC Technology Conference, TPCTC 2018*. 2019, pp. 58–70 (cit. on p. 17).

[225] Yuzhe Tang and Bugra Gedik. "Autopipelining for data stream processing." In: *IEEE Transactions on Parallel and Distributed Systems* 24.12 (2012), pp. 2344–2354 (cit. on pp. 6, 88, 93, 95, 102, 124, 125, 132).

[226] Muhammad Adnan Tariq, Boris Koldehofe, Sukanya Bhowmik, and Kurt Rothermel. "PLEROMA: A SDN-based high performance publish/subscribe middleware." In: *Proceedings of the 15th International Middleware Conference*. 2014, pp. 217–228 (cit. on p. 36).

[227] Zois-Gerasimos Tasoulas, Georgios Zervakis, Iraklis Anagnostopoulos, Hussam Amrouch, and Jörg Henkel. "Weight-oriented approximation for energy-efficient neural network inference accelerators." In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 67.12 (2020), pp. 4670–4683 (cit. on p. 135).

[228] Khin Me Me Thein. "Apache kafka: Next generation distributed messaging system." In: *International Journal of Scientific Engineering and Technology Research* 3.47 (2014), pp. 9478–9483 (cit. on p. 53).

[229] Lasse Thostrup, Jan Skrzypczak, Matthias Jasny, Tobias Ziegler, and Carsten Binnig. "DFI: The Data Flow Interface for High-Speed Networks." In: *Proceeding of the International Conference on Management of Data*. 2021, pp. 1825–1837 (cit. on p. 36).

[230] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. “Domain randomization for transferring deep neural networks from simulation to the real world.” In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* 2017, pp. 23–30 (cit. on p. 54).

[231] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. “Domain randomization for transferring deep neural networks from simulation to the real world.” In: *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS)* (cit. on p. 54).

[232] Theodoros Toliopoulos and Anastasios Gounaris. “Adaptive distributed partitioning in apache flink.” In: *Proceedings of the IEEE 36th International Conference on Data Engineering Workshops (ICDEW).* 2020, pp. 127–132 (cit. on p. 35).

[233] Ralf Tönjes, P Barnaghi, M Ali, A Mileo, M Hauswirth, F Ganz, S Ganea, B Kjærgaard, D Kuemper, Septimiu Nechifor, et al. “Real time iot stream processing and large-scale data analytics for smart city applications.” In: *poster session, European Conference on Networks and Communications.* 2014, p. 10 (cit. on p. 30).

[234] Donato Toppeta. *Smart Cities, not only new Research Papers but also exciting forecast!* `http://ict4green.wordpress.com/2012/02/11/smart-cities-not-only-newresearch-papers-but-also-exiting-forecast/`. [Online; accessed 25-06-2024]. 2012 (cit. on p. 9).

[235] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. “Storm@ twitter.” In: *Proceedings of the ACM SIGMOD international conference on Management of data.* 2014, pp. 147–156 (cit. on p. 29).

[236] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh Mittal, and Dmitriy Ryaboy. “Storm@twitter.” In: *Proceedings of the ACM International Conference on Management of Data (SIGMOD).* 2014, pp. 147–156 (cit. on pp. 43, 47, 133).

[237] Pete Tucker, Kristin Tufte, Vassilis Papadimos, and David Maier. “Nexmark–a benchmark for queries over data streams (draft).” In: *Technical report, Technical Report. Technical report* (2008) (cit. on p. 58).

[238] Giselle Van Dongen and Dirk Van den Poel. “Evaluation of stream processing frameworks.” In: *IEEE Transactions on Parallel and Distributed Systems* 31.8 (2020), pp. 1845–1858 (cit. on pp. 2, 16, 44, 45).

[239] Jana Vatter, Ruben Mayer, and Hans-Arno Jacobsen. “The evolution of distributed systems for graph neural networks and their origin in graph processing and deep learning: A survey.” In: *ACM Computing Surveys* 56.1 (2023), pp. 1–37 (cit. on pp. 18, 22).

[240] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. “Apache hadoop yarn: Yet another resource negotiator.” In: *Proceedings of the 4th annual Symposium on Cloud Computing.* 2013, pp. 1–16 (cit. on p. 98).

[241] Swagath Venkataramani, Vijayalakshmi Srinivasan, Wei Wang, Sanchari Sen, Jintao Zhang, Ankur Agrawal, Monodeep Kar, Shubham Jain, Alberto Mannari, Hoang Tran, et al. “RaPiD: AI accelerator for ultra-low precision training and inference.” In: *Proceedings of the ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA).* 2021, pp. 153–166 (cit. on p. 135).

[242] Uri Verner, Assaf Schuster, and Mark Silberstein. “Processing Data Streams with Hard Real-Time Constraints on Heterogeneous Systems.” In: *Proceedings of the International Conference on Supercomputing*. 2011, pp. 120–129 (cit. on p. 36).

[243] Juan J Villalobos, Ivan Rodero, and Manish Parashar. “An unsupervised approach for online detection and mitigation of high-rate DDoS attacks based on an in-memory distributed graph using streaming data and analytics.” In: *Proceedings of the 4th IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*. 2017, pp. 103–112 (cit. on p. 23).

[244] Adriano Vogel, Dalvan Griebler, Marco Danelutto, and Luiz Gustavo Fernandes. “Self-adaptation on parallel stream processing: A systematic review.” In: *Concurrency and Computation: Practice and Experience* 34.6 (2022), e6759 (cit. on pp. 16, 19, 23).

[245] Chang Wang and Sridhar Mahadevan. “Heterogeneous domain adaptation using manifold alignment.” In: *IJCAI proceedings-international joint conference on artificial intelligence*. Vol. 22. 1. 2011, p. 1541 (cit. on p. 25).

[246] Ching-Hao Wang, Kang-Yang Huang, Yi Yao, Jun-Cheng Chen, Hong-Han Shuai, and Wen-Huang Cheng. “Lightweight Deep Learning: An Overview.” In: *IEEE Consumer Electronics Magazine* 13.4 (2024), pp. 51–64 (cit. on p. 135).

[247] Di Wang, Elke A Rundensteiner, Han Wang, and Richard T Ellison III. “Active complex event processing: applications in real-time health care.” In: *Proceedings of the VLDB Endowment* 3.1-2 (2010), pp. 1545–1548 (cit. on pp. 1, 29).

[248] Dong Wang and Thomas Fang Zheng. “Transfer learning for speech and language processing.” In: *Proceedings of the Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*. 2015, pp. 1225–1237 (cit. on pp. 24, 25).

[249] Fei Wang, James Decker, Xilun Wu, Gregory Essertel, and Tiark Rompf. “Backpropagation with callbacks: Foundations for efficient and expressive differentiable programming.” In: vol. 31. 2018 (cit. on pp. 4, 22).

[250] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, et al. “Bigdatabench: A big data benchmark suite from internet services.” In: *Proceedings of the IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 2014, pp. 488–499 (cit. on pp. 2, 16, 44, 45).

[251] Yangjun Wang et al. “Stream processing systems benchmark: Streambench.” MA thesis. 2016 (cit. on p. 16).

[252] Pallavi Wankhede, Minaiy Talati, and Rutuja Chinchamalatpure. “Comparative study of cloud platforms-microsoft azure, google cloud platform and amazon EC2.” In: *Journal of Engineering Research and Applied Science* 5.02 (2020), pp. 60–64 (cit. on p. 31).

[253] Johannes Wehrstein, Benjamin Hilprecht, Benjamin Olt, Manisha Luthra, and Carsten Binnig. “The Case for Multi-Task Zero-Shot Learning for Databases.” In: *Proceeding of the 4th International Workshop on Applied AI for Database Systems and Applications (AIDB) colocated with VLDB 2022*. 2022, pp. 1–8 (cit. on p. 134).

[254] Pascal Weisenburger, Manisha Luthra, Boris Koldehofe, and Guido Salvaneschi. “Quality-Aware Runtime Adaptation in Complex Event Processing.” In: *Proceedings of the IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 2017, pp. 140–151 (cit. on pp. 2, 23, 47).

[255] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. “A survey of transfer learning.” In: *Journal of Big data* 3 (2016), pp. 1–40 (cit. on p. 24).

[256] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga, Jinshi Huang, Charles Bai, et al. “Sustainable ai: Environmental implications, challenges and opportunities.” In: *Machine Learning and Systems* 4 (2022), pp. 795–813 (cit. on p. 134).

[257] Song Wu, Die Hu, Shadi Ibrahim, Hai Jin, Jiang Xiao, Fei Chen, and Haikun Liu. “When FPGA-accelerator meets stream data processing in the edge.” In: *Proceedings of the IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. 2019, pp. 1818–1829 (cit. on p. 133).

[258] Ziniu Wu, Ryan Marcus, Zhengchun Liu, Parimarjan Negi, Vikram Nathan, Pascal Pfeil, Gaurav Saxena, Mohammad Rahman, Balakrishnan Narayanaswamy, and Tim Kraska. “Stage: Query Execution Time Prediction in Amazon Redshift.” In: *Proceedings of the Companion of the International Conference on Management of Data*. 2024, pp. 280–294 (cit. on pp. 108, 109, 133).

[259] Jielong Xu, Jian Tang, Zhiyuan Xu, Chengxiang Yin, Kevin Kwiat, and Charles Kamhoua. “A deep recurrent neural network based predictive control framework for reliable distributed stream data processing.” In: *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2019, pp. 262–272 (cit. on p. 21).

[260] Le Xu, Boyang Peng, and Indranil Gupta. “Stela: Enabling stream processing systems to scale-in and scale-out on-demand.” In: *Proceedings of the IEEE International Conference on Cloud Engineering (IC2E)*. 2016, pp. 22–31 (cit. on p. 19).

[261] Ting-Bing Xu, Peipei Yang, Xu-Yao Zhang, and Cheng-Lin Liu. “LightweightNet: Toward fast and lightweight convolutional neural networks via architecture distillation.” In: *Pattern Recognition* 88 (2019), pp. 272–284 (cit. on p. 135).

[262] Kyung-A Yoon, Oh-Sung Kwon, and Doo-Hwan Bae. “An approach to outlier detection of software measurement data using the k-means clustering method.” In: *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 2007, pp. 443–445 (cit. on pp. 57, 64).

[263] Nikos Zacheilas, Vana Kalogeraki, Nikolas Zygouras, Nikolaos Panagiotou, and Dimitrios Gunopulos. “Elastic complex event processing exploiting prediction.” In: *Proceedings of the IEEE International Conference on Big Data (BigData)*. 2015, pp. 213–222 (cit. on p. 15).

[264] Eleni Zapridou, Ioannis Mytilinis, and Anastasia Ailamaki. “Dalton: Learned Partitioning for Distributed Data Streams.” In: *Proceeding of the VLDB Endowment* 16.3 (2022), pp. 491–504 (cit. on pp. 23, 72, 73, 134).

[265] Eleni Zapridou, Ioannis Mytilinis, and Anastasia Ailamaki. “Dalton: Learned Partitioning for Distributed Data Streams.” In: *Proceedings of the VLDB Endowment* 16.3 (2022), pp. 491–504 (cit. on pp. 44–46, 131).

[266] Steffen Zeuch, Xenofon Chatziliadis, Ankit Chaudhary, Dimitrios Giouroukis, Philipp M Grulich, Dwi Prasetyo Adi Nugroho, Ariane Ziehn, and Volker Mark. “NebulaStream: Data Management for the Internet of Things.” In: *Datenbank-Spektrum* 22.2 (2022), pp. 131–141 (cit. on p. 133).

[267] Steffen Zeuch, Bonaventura Del Monte, Jeyhun Karimov, Clemens Lutz, Manuel Renz, Jonas Traub, Sebastian Breß, Tilmann Rabl, and Volker Markl. "Analyzing efficient stream processing on modern hardware." In: *Proceedings of the VLDB Endowment* 12.5 (2019), pp. 516–530 (cit. on pp. 1, 2, 17, 44, 45, 57).

[268] Feng Zhang, Lin Yang, Shuhao Zhang, Bingsheng He, Wei Lu, and Xiaoyong Du. "{FineStream}:{Fine-Grained}{Window-Based} stream processing on {CPU-GPU} integrated architectures." In: *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC 20)*. 2020, pp. 633–647 (cit. on pp. 36, 133).

[269] Liang Zhang, Wenli Zheng, Chao Li, Yao Shen, and Minyi Guo. "Autrascale: an automated and transfer learning solution for streaming system auto-scaling." In: *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2021, pp. 912–921 (cit. on p. 24).

[270] Shuhao Zhang, Yancan Mao, Jiong He, Philipp M Grulich, Steffen Zeuch, Bingsheng He, Richard TB Ma, and Volker Markl. "Parallelizing intra-window join on multicores: An experimental study." In: *Proceedings of the International Conference on Management of Data*. 2021, pp. 2089–2101 (cit. on p. 133).

[271] Xiaochen Angela Zhang and Raluca Cozma. "Risk sharing on Twitter: Social amplification and attenuation of risk in the early stages of the COVID-19 pandemic." In: *Computers in Human Behavior* 126 (2022), p. 106983 (cit. on p. 29).

[272] Yongpeng Zhang and Frank Mueller. "Gstream: A general-purpose data streaming framework on gpu clusters." In: *Proceedings of the International Conference on Parallel Processing*. 2011, pp. 245–254 (cit. on p. 133).

[273] Joey Zhou, Sinno Pan, Ivor Tsang, and Yan Yan. "Hybrid heterogeneous transfer learning through deep learning." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2014, pp. 1–28 (cit. on p. 25).

[274] Yin Zhu, Yuqiang Chen, Zhongqi Lu, Sinno Pan, Gui-Rong Xue, Yong Yu, and Qiang Yang. "Heterogeneous transfer learning for image classification." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 25. 1. 2011, pp. 1304–1309 (cit. on p. 25).

[275] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. "A comprehensive survey on transfer learning." In: *Proceedings of the IEEE* 109.1 (2020), pp. 43–76 (cit. on p. 24).

[276] Tobias Ziegler, Dwarakanandan Bindiganavile Mohan, Viktor Leis, and Carsten Binnig. "EFA: A Viable Alternative to RDMA over InfiniBand for DBMSs?" In: *Proceeding of the 18th International Workshop on Data Management on New Hardware*. 2022 (cit. on p. 36).

[277] Tobias Ziegler, Carsten Binnig, and Viktor Leis. "ScaleStore: A Fast and Cost-Efficient Storage Engine using DRAM, NVMe, and RDMA." In: *Proceeding of the International Conference on Management of Data*. 2022, pp. 685–699 (cit. on p. 36).

All web pages cited in this work have been checked in June 2024. However, due to the dynamic nature of the World Wide Web, their long-term availability cannot be guaranteed.

Acronyms

DAG Directed Acyclic Graph. 11, 40, 60, 77, 109

DNNs Deep Neural Networks. 21, 22, 25

DSP Distributed Stream Processing. v, vii, viii, xiv–xvii, 1–7, 9–25, 27, 29–33, 36–38, 40–50, 52, 53, 57, 58, 67–75, 78–82, 89, 95, 96, 98, 102, 105, 110, 113, 120, 122, 123, 125–127, 129–135, 161–163, 166, 181, 182

GNN Graph Neural Network. 22, 23, 74, 77, 83, 109, 131, 182

IoT Internet of Things. 9, 32, 36

ML machine learning. 15, 16, 18–25, 131, 134, 135

MLP Multi-Layer Perceptron. 78, 83, 85, 86

Non-ML Non-machine learning. 15, 18

PDSP parallel and distributed stream processing. 15, 17, 20, 23, 28, 40–43, 50, 57, 70, 104–107, 129, 130, 133

PQP parallel query structures. 51–56, 68, 70, 74, 76–79, 81, 83–88, 92, 99, 100, 103–109, 111–117, 119–123, 126, 127, 162, 164–171, 173–181, 183

PSP parallel stream processing. 12, 13

QoS Quality of Service. xv, 3, 11, 31, 32, 41, 47, 49, 50

SP Stream Processing. 52, 105, 106, 108

SPS Stream Processing System. 51, 103

SUT System under Test. 51–58, 67–69, 98–102, 127, 163, 164, 166, 168

UDOs user-defined operators. 103–105, 107

WUI Web User Interface. 52, 53, 69, 163, 165–167

A

Appendix

A.1 Supplementary Material to Chapter 4

In the following, we provide a difference between parallel and sequential distributed stream processing in Section A.1.1 followed by an overview of PDSP-Bench components in Section A.1.2. Additionally, we present web user interface (WUI) of PDSP-Bench for performance benchmarking based on user inputs in Section A.1.3. Finally, Section A.1.4 shows insights into the performance evaluation of PDSP-Bench for other performance metrics like resource utilization.

A.1.1 *Parallel and Sequential Distributed Stream Processing*

PDSP for handling higher workload

Multiple instances of operator in PDSP

Parallel and sequential stream processing in the context of DSP systems highlight *how data is processed across distributed systems*. In DSP systems, data streams can be processed by *single or multiple instances* of operators which are placed on a single machine or distributed across multiple cloud resources. It is essential to understand the distinctions between parallel and sequential distributed stream processing for designing DSP systems that efficiently handle data at scale while meeting specific performance requirements of applications. For instance, in parallel distributed stream processing (PDSP), data streams are partitioned and processed concurrently by multiple instances of operators in DSP systems, leveraging the combined computational resources to process large volumes of data simultaneously, aiming to improve throughput and reduce latency. While, *sequential distributed stream processing* is a traditional method involving each operator in an operator graph sequentially processing data streams at a time, distributed across cloud resources. The fundamental difference between parallel and sequential distributed stream processing is presented in Table 10 to clear the concept used in this dissertation.

Feature	Parallel Distributed Stream Processing	Sequential Distributed Stream Processing
Definition	Processes data streams concurrently by multiple instances of operators.	Processes data streams in sequence of operators, one data at a time, across nodes.
Scalability	High, with the ability to add more nodes for processing larger data volumes.	Moderate, scalability is limited by the sequential processing capacity of nodes.
Processing Time	Reduced due to concurrent processing across nodes.	Longer due to the sequential processing of data elements.
Throughput	High, ideal for applications requiring fast data processing.	Lower, suited for less time-sensitive tasks.
Complexity	Higher, due to the need for managing concurrency and data partitioning.	Lower, with simpler data flow management.
Use Cases	Suited for real-time analytics and applications needing quick processing.	Ideal for batch processing and tasks where data sequence is paramount.
Resource Utilization	Efficient, leveraging full computational power across nodes.	Potentially less efficient, with a risk of underutilization of resources.
Query Plan	Involves complex planning for data partitioning and task distribution.	Simpler query plans, with a focus on the sequence of operations across nodes.
Data Stream	Handles high-volume, high-velocity data streams efficiently.	Better suited for lower-volume, less time-critical data streams.
Resource	Requires a robust network and computational resources to manage parallel tasks.	Can operate with less demanding resource due to sequential processing.

Table 10: Differences in parallel and sequential stream processing related to important features of DSP systems such as scalability, latency, throughput, etc. [2].

A.1.2 *Implementation Details of* PDSP-Bench

This section provides the process flow between components of PDSP-Bench for benchmarking, including resource provisioning using CloudLab, executing PQP for benchmarking, and collecting benchmarking data for further training of learned cost models such as [1].

Resource provisioning and Cluster Deployment

Once the cloud infrastructure is set up in CloudLab, PDSP-Bench needs to provision resources on which System Under Test (SUT), i.e., DSP systems need to be benchmarked for parallel and distributed stream processing. For

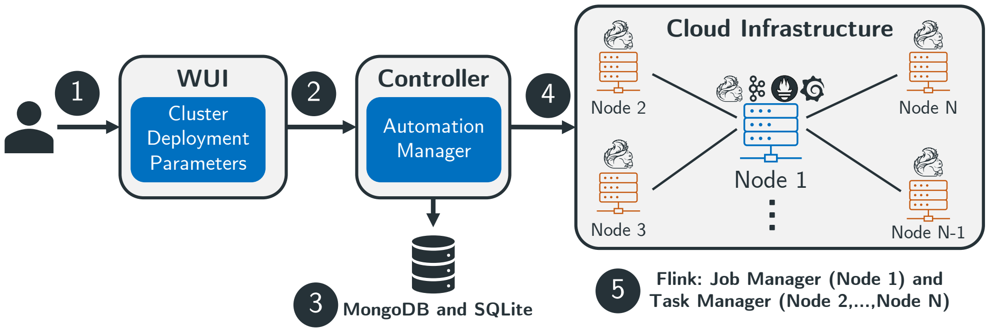


Figure 52: Resource provisioning and Apache Flink cluster setup on CloudLab cluster nodes. ① User provides cluster setup parameters which are ② forwarded to `controller`. ③ these user's setup parameters are stored in *MongoDB* and *SQLite* databases. ④ `Controller's automation manager` initiates the cluster deployment process where ⑤ the Flink job manager is set on *Node 1* and the rest of the other nodes can be used as Task managers, which are responsible for executing the tasks based on the query. Data producer *Kafka* is to simulate data streams for various applications as well as *Prometheus* and *Grafana* are used for monitoring. User inputs are stored in *MongoDB* and *SQLite* databases [2].

this, users can provide details of CloudLab cluster nodes or VM machines they want to set up DSP, e.g., *hostname*, and *CloudLab account username*. These details can be provided via a web user interface (WUI) (cf. Section A.1.3) of PDSP-Bench as presented in Figure 52. The `WUI` sends this information via an *HTTP POST* request to the *Django* backend, where the Infra module saves the node[8] details in an SQLite database.

CloudLab for setting distributed environment

With the cluster nodes addresses saved, the next step is setting up the SUT like *Apache Flink*, monitoring tools such *Grafana* and *Prometheus*, and data stream producing through *Kafka* to simulate real-time parallel and distributed environment for stream processing. These steps can be easily set again through simple steps at `WUI` of PDSP-Bench where user *Create Cluster* in the `WUI` by providing cluster details such as the number of task slots per task manager node and the number of task manager nodes for executing queries as presented in Figure. The `WUI` sends configuration details via *HTTP request* to the `controller` Section 4.5 via backend API (application programming interface).

Prometheus and Grafana for performance monitoring

The `controller` processes this configuration, updating the database and setting *Ansible* variables accordingly. It then executes the cluster setup Ansible playbook, which runs a series of *Ansible* tasks. Ansible communicates with the CloudLab nodes to create an Apache Flink distributed environment where Apache Flink's job manager, data producer - Kafka, and monitoring tools - Grafana and Prometheus, are set up on one of the cluster nodes (by

Cluster setup using controller

[8] CloudLab clusters consist of multiple nodes. Nodes refer to a single physical machine within a cluster.

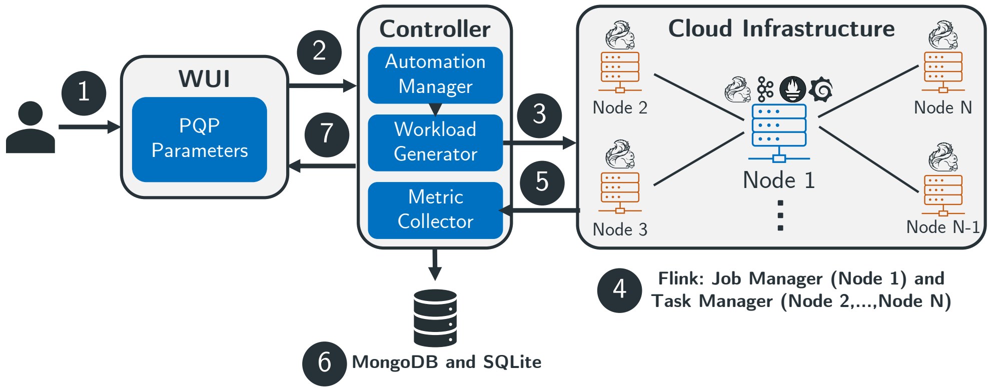


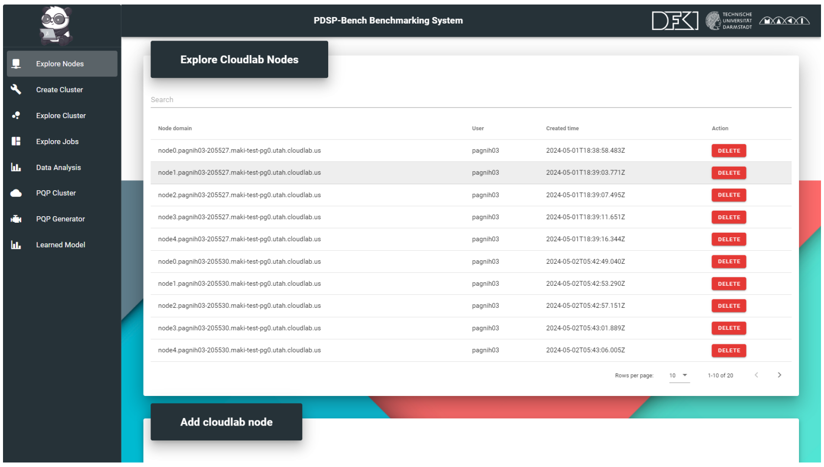
Figure 53: PQP execution on SUT followed by real-time and historical performance analysis. ① User provides parameters for PQP from real-world and synthetic applications. ② `Automation manager` collects these parameters and ③ executes these queries on Flink clusters. ④ Flink uses a job manager to assign these tasks to different task managers to execute the query. ⑤ Monitoring tools like Prometheus and Grafana report the real-time performance to ⑥ `Metric collectors` to store provided configuration and corresponding performance in a database for historical analysis. ⑦ `Automation manager` fetches this information to visualize the real-time performance or historical analysis for user [2].

default node 0), referred to as the master node. Other remaining nodes in the clusters are set up for task managers. At this stage, the benchmarking system is ready to execute PQP for different parallel query structures on Apache Flink jobs to benchmark its performance under varying workloads and collect performance benchmarking data for further analysis and training of the learned cost models.

Benchmarking using PDSP-Bench

User-defined parameters to set up benchmarking process

After setting up the clusters for benchmarking, the next step involves executing a stream processing job to SUT such as Apache Flink. Figure 53 illustrates the communication flow for job submission and performance metric collection. For this, users can use PQP from both real-world and synthetic applications by providing various query and data stream-specific parameters such as event rate, window size and slide, parallelism of each job operator, the number of job iterations, and the duration for each run. These parameters are forwarded to PDSP-Bench `controller` via HTTP POST request to trigger `Automation manager` to connect to the master node, submitting the job to the Flink Job Manager with the user-defined parameters. After the job execution completes, monitoring tools collect the performance metrics on the master node and store the performance data from *Prometheus* to locally on the master node or *MongoDB* and *SQLite* databases. This process ensures that the stream processing job is executed with the specified configurations and the performance metrics are accurately collected for analysis.



Node domain	User	Created time	Action
node0.pagnih03-205527.maki-test-pg0.utah.cloudlab.us	pagnih03	2024-05-01T18:38:58.483Z	DELETE
node1.pagnih03-205527.maki-test-pg0.utah.cloudlab.us	pagnih03	2024-05-01T18:39:03.771Z	DELETE
node2.pagnih03-205527.maki-test-pg0.utah.cloudlab.us	pagnih03	2024-05-01T18:39:07.495Z	DELETE
node3.pagnih03-205527.maki-test-pg0.utah.cloudlab.us	pagnih03	2024-05-01T18:39:11.651Z	DELETE
node4.pagnih03-205527.maki-test-pg0.utah.cloudlab.us	pagnih03	2024-05-01T18:39:16.344Z	DELETE
node0.pagnih03-205530.maki-test-pg0.utah.cloudlab.us	pagnih03	2024-05-02T05:42:49.040Z	DELETE
node1.pagnih03-205530.maki-test-pg0.utah.cloudlab.us	pagnih03	2024-05-02T05:42:53.290Z	DELETE
node2.pagnih03-205530.maki-test-pg0.utah.cloudlab.us	pagnih03	2024-05-02T05:42:57.151Z	DELETE
node3.pagnih03-205530.maki-test-pg0.utah.cloudlab.us	pagnih03	2024-05-02T05:43:01.889Z	DELETE
node4.pagnih03-205530.maki-test-pg0.utah.cloudlab.us	pagnih03	2024-05-02T05:43:06.005Z	DELETE

Rows per page: 10 1-10 of 20

Figure 54: Resource provisioning through GENI and CloudLab infrastructure [2].

Real-time Performance Analysis

Subsequently, PDSP-Bench offers capabilities for real-time visualization of currently executing PQP performance and comparison of previously executed PQP performance as shown in Figure 53.

For real-time performance visualization, PDSP-Bench uses the REST API provided by Flink to obtain performance metrics. The `controller` processes, structures, and cleans this data, then visualizes it using D3.js plots on the `WUI`, providing frequently updated live graphs.

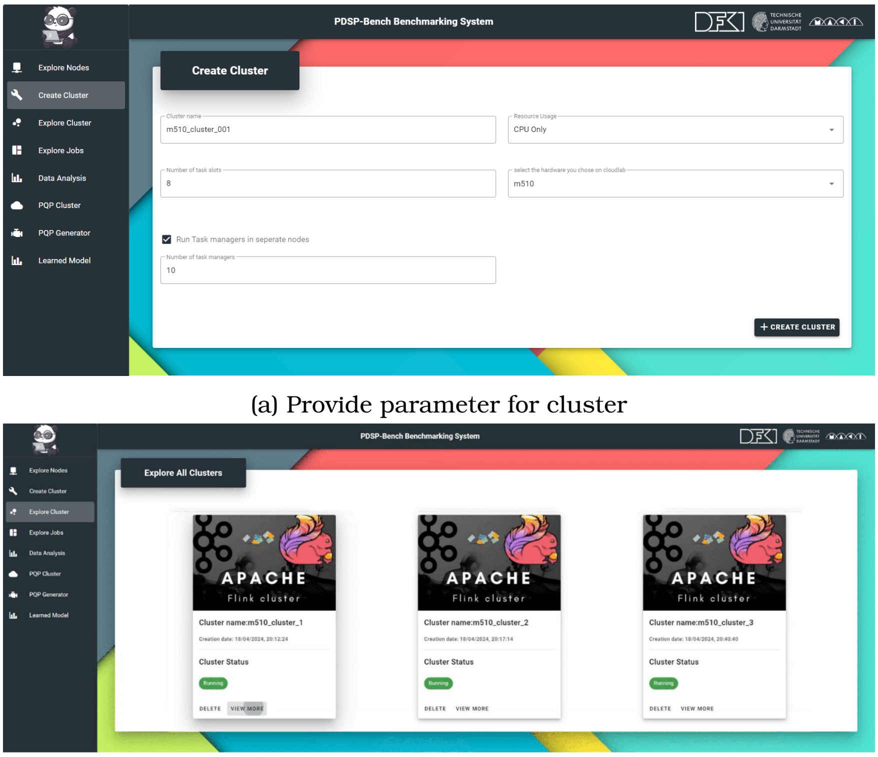
Performance analysis and visualization on `WUI`

The `controller` uses the performance analytics module to visualize historical performance metrics and compare data from recently completed jobs. This module collects user input on jobs, operators, and metrics to be compared. It fetches the related metrics from the Prometheus API, structures the data based on comparison choices, and visualizes it on the `WUI`.

A.1.3 *Navigating Web User Interface of* PDSP-Bench

Create distributed cloud environment

This section delves into the visual representation of the `WUI` in PDSP-Bench, outlining the steps involved in the benchmarking process. After creating resource nodes on CloudLab and starting the PDSP-Bench `controller`, users should gather the hostnames of these nodes. With this information in hand, the next step is to access the PDSP-Bench frontend, i.e., `WUI`, through a web browser to configure and manage the benchmarking tasks as presented in Figure 54.



Create Cluster

Cluster name: m510_cluster_001

Resource Usage: CPU Only

Number of task slots: 8

select the hardware you chose on cloudlab: m510

☑ Run Task managers in seperate nodes

Number of task managers: 10

(a) Provide parameter for cluster

Explore All Clusters

Cluster name:m510_cluster_1
Creation date: 18/04/2024, 20:12:24
Cluster Status
Running

Cluster name:m510_cluster_2
Creation date: 18/04/2024, 20:17:14
Cluster Status
Running

Cluster name:m510_cluster_3
Creation date: 18/04/2024, 20:40:40
Cluster Status
Running

(b) Explore already created clusters

Figure 55: Configuring DSP system as SUT for performance benchmarking [2].

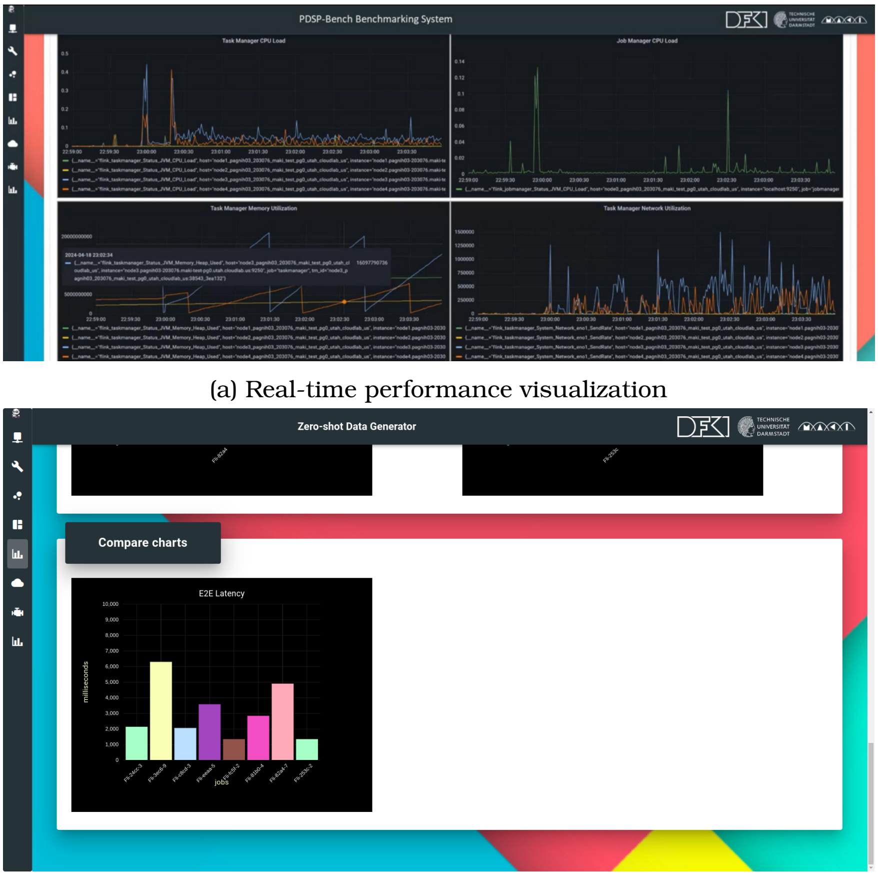
Set up nodes for SUT deployment

To begin, navigate to the `Explore Node` tab from the navigation bar on the left of the WUI. This tab is designed to manage and interact with the nodes from your CloudLab cluster. Here, users are required to input the hostnames of all the nodes along with their CloudLab usernames. This step is essential for the system to recognize and effectively manage the nodes.

By completing these tasks, users ensure that their nodes are properly configured and ready for benchmarking. The WUI in PDSP-Bench provides a streamlined and user-friendly interface to facilitate the setup, making it easier to utilize the tool's full capabilities for performance evaluation and analysis.

Create cluster for DSP

After adding the CloudLab nodes, users can proceed by navigating to the `Create Cluster` tab from the navigation menu to create a cluster with the necessary parameters, as shown in (cf. Figure 55a). Users can create multiple clusters by dividing the number of CloudLab nodes from the previous cluster configurations. Alternatively, users can explore already created clusters to start benchmarking the SUT for various PQP scenarios, both synthetic and real-world, as illustrated in Figure 55b.



(a) Real-time performance visualization

(b) Performance analysis

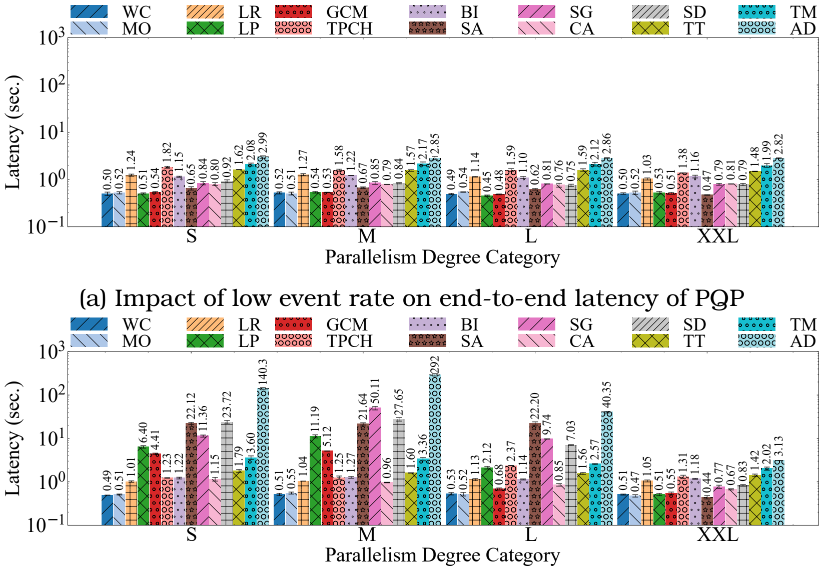
Figure 56: Performance visualization of PQP from real-world and synthetic applications [2].

To do this, users should navigate to the `Explore Cluster` tab from the navigation menu. In this tab, they can click on `View More` and then the `Provide Jobs` tab to execute queries with various workloads and query parameters. This step allows users to specify the parameters for the benchmarking tasks, including different event rates, parallelism degrees, execution times, and the number of iterations. The user-friendly interface of the `WUI` ensures that users can efficiently manage and monitor their benchmarking processes, facilitating a thorough evaluation of performance across different cluster configurations.

User-defined parameters to execute query

To monitor the real-time performance of executing jobs, navigate to the `Explore Jobs` tab from the navigation menu. Within this tab, you will find a list of currently running jobs. Click on the `View More` button next to each job to visualize its real-time performance metrics, as illustrated in Figure 56a. This feature allows you to track the ongoing performance and efficiency of your stream processing tasks, providing immediate insights into operational metrics.

Monitor and visualization of performance



(a) Impact of low event rate on end-to-end latency of PQP

(b) Impact of increasing event rate on end-to-end latency of PQP

Figure 57: Impact of parallelism degree on PQP performance from real-world applications. The analysis shows distinct performance behaviors for varying event rate 100 (top) and 100k (bottom) [2, 4].

Generate and benchmark data for ML

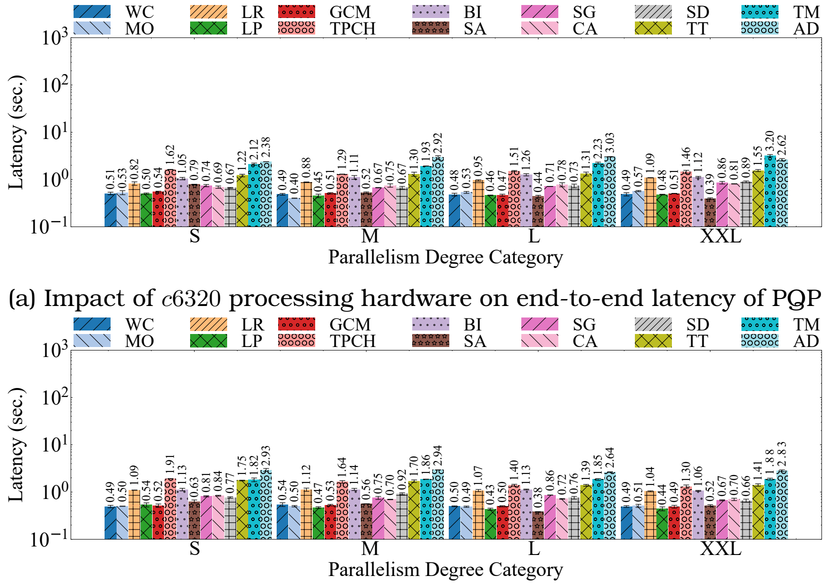
For analyzing and comparing historical performance metrics, navigate to the `Data Analytics` tab from the navigation menu. This tab is designed to facilitate a detailed comparison of performance data from jobs that have already been executed. As shown in Figure 56b, you can select various jobs, operators, and metrics to visualize and compare their historical performance. This functionality is particularly useful for identifying trends, bottlenecks, and performance variations over time.

A.1.4 *Additional Insight on Performance Benchmarking*

In this section, we discuss additional results of performance benchmarking of SUT, i.e., Apache Flink. We provide our interesting observation on the impact of performance PQP for different event rates and resource diversity.

Impact of Event Rate on Performance

We benchmark real-world applications for various workload and resource configurations, including different event rates parallelism categories, and analyze their impact on performance (cf. Table 7). In Section 6.2, we discuss the impact of parallelism and workload diversity on performance, where we discuss



(a) Impact of $c6320$ processing hardware on end-to-end latency of PQP

(b) Impact of $c6525\_25$ processing hardware on end-to-end latency of PQP

Figure 58: Impact of heterogeneous hardware on PQP performance from real-world applications. The analysis shows distinct performance behaviors of event rate for different hardware $c6320$ (top) and $c6525\_25g$ (bottom) [2, 4].

higher event rates resulting in backpressure, which can be handled by increasing parallelism.

Parallelism is not helpful for low event rate

In Figure 57, we further provide our observation where the lower event has zero or negligible effect on performance improvement when parallelism is increased. For instance, in Figure 57a shows the performance for different parallelism categories when the event for different PQP is relatively low, i.e., 100 events per second. In this scenario, we observe that the increasing parallelism has resulted in minimal improvement in performance as this event rate can be easily tackled with low parallelism for all the real-world applications.

Higher event rate leads to backpressure

When we further increase the event rate, i.e., 100k followed by increasing the parallelism to improve the performance, then we observe distinct behavior in performance as presented in Figure 57b. For instance, PQP with the less data-intensive task has the negligible effect of increasing parallelism such as `WC, MO, LR` while PQP with the data-intensive task have degraded performance for lower parallelism, i.e., S, M, L and the performance improves significantly as the parallelism is more than 128, i.e., XXL such as `AD, SA`.

Impact of Heterogeneous Hardware on Event Rate

In Section 6.2, we discuss hardware diversity and its influences on performance. We observe that higher processing capabilities do not always lead to improved performance, as it depends on various query and operator parameters. Increasing processing capability or using heterogeneous configurations can introduce additional overhead for state management and synchronization, potentially degrading performance.

Parallelism handles backpressure...

...and improves performance.

Figure 58 provides our observation for different heterogeneous hardware configurations with higher processing capabilities. Figure 58a shows performance for varying parallelism categories on hardware $c6320$ with 28 processing cores, while Figure 58a shows performance on hardware $c6525\_25g$ with 16 cores for a minimal event rate, i.e., 100 event/sec. Neither increasing parallelism nor enhancing processing capabilities consistently improves performance, showing either slight or negligible improvements. Notably, similar performance levels were achieved with lower processing hardware, such as $m510$, using minimal parallelism categories (cf. Figure 57). Such performance understanding is crucial, especially in distributed stream processing environments where cloud resources have diverse hardware configurations. Without performance insights, resources may be under or over-utilized, resulting in suboptimal performance.

Impact of Parallelism on Resource Utilization

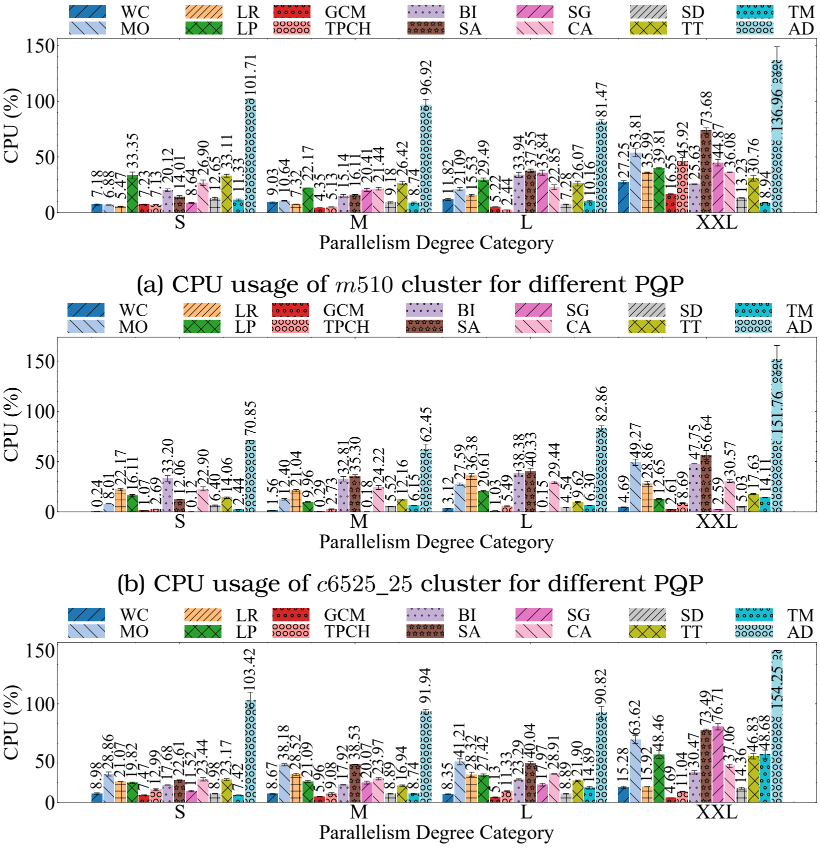
Better resource utilization...

We perform a performance analysis to understand the impact of end-to-end latency and resource utilization (percentage of CPU usage) with different parallelism categories and diverse hardware configurations. Figure 59 presents percentage CPU usage for varying hardware resources with different parallelism categories. The primary aim of this evaluation is to understand how resource usage gets impacted by varying processing capabilities and increasing parallelism categories. We present results for the event rate of $1M$ event/second to put backpressure to utilize the benefits of parallelism efficiently.

...for higher event rate.

The evaluation provides various exciting insights. First, it can be noticed that with increasing parallelism categories, the percentage of CPU usage increases as well because the increase in parallelism requires more cores from the processing hardware to create multiple instances to process data-intensive tasks for real-world applications. For instance, higher parallelism resulted in improved performance, i.e., end-to-end latency (cf. Figure 35). This is evident in Figure 59a that the higher parallelism resulted in more utilization of CPU cores, which resulted in improved performance for PQP from most of the real-world applications such as AD, SG, CA.

One key finding with practical implications is that increased processing capability and hardware diversity lead to increased CPU usage. This suggests that as processing capabilities and hardware diversity increase, so does the demand for CPU resources. However, it is important to note that not all ap-



(a) CPU usage of $m510$ cluster for different PQP

(b) CPU usage of $c6525\_25$ cluster for different PQP

(c) CPU usage of $c6320$ cluster for different PQP

Figure 59: Resource utilization in terms of CPU usage in percentage for varying parallelism degrees. It presents the impact of hardware processing capabilities $m510$ (top), $c6525\_25g$ (middle) and $c6320$ (bottom) with increasing parallelism degrees for event rate $1M$ event/sec [2, 4].

plications benefit from this increase in resources, as seen in Figures 59b and 59c. For instance, less data-intensive real-world applications such as WC, SG, LR have shown consistent CPU usage across increasing parallelism categories and hardware diversity. This reiterates the main point of our analysis, which is that the impact of parallelism and hardware diversity on CPU usage is not uniform across all applications. This could be due to the fact that these queries are not data intensive and, therefore, have the same CPU usage or minimal CPU usage as they are less data-intensive and may not require over-utilization of resources in all possible scenarios.

Not all applications need high processing resources

A.2 Supplementary Material to Chapter 5

In the following, Section A.2.1 provides the additional conceptual information about transfer learning followed by additional insights into the performance evaluation results using naive training strategy ZT-Random in Section A.2.2. Additionally, we present the evaluation of different message-passing schemes for the training model in Section A.2.3 followed by the distribution of training data in Section A.2.5.

A.2.1 *Zero-shot, One-shot and Few-shot Learning*

Zero-shot, one-shot, and few-shot learning [1, 3] are transfer learning techniques that allow machine learning models to recognize patterns, make predictions, or perform tasks with minimal exposure to labeled data. These techniques are particularly useful when obtaining labeled data is costly or impractical.

Train and test data are disjoint

Zero-Shot Learning (ZSL): It is a method where a model is trained to recognize and handle classes it has never explicitly seen during training. It typically relies on understanding some form of semantic relationship between classes known during training and those not seen (e.g., through attributes or embeddings). Using zero-shot learning, the system could be trained to recognize certain patterns based on their characteristics defined through metadata or embeddings. Even if a specific type of patterns has never been encountered during the training phase, the system could identify it based on learned semantic characteristics common to other types of patterns. For instance, in the context of ZeroTune, the model is trained on query of different patterns given by its input data like data source types, operator types and its placement. This way it learns the semantic relationship of the performance costs of a query through these trained features so that it can derive cost correlations even to unseen query characteristics during inference.

Model sees one or few test data

One-Shot Learning (OSL): It involves training a model in such a way that it learns to recognize or categorize objects from seeing just one, or a few, examples. It's often used in classification tasks where only a single example of each class is available to the model during training. In a one-shot learning scenario, a stream processing system could be set up to classify types of system logs or error messages by seeing just one example of each type. This capability is useful for rapidly evolving systems where new types of logs or errors may frequently appear, and it is necessary to quickly categorize them without waiting for large amounts of labeled data.

Trained Model	Query	Q-error (Latency)		Q-error (Tpt)	
ZeroTune-ZT-Random	**Structure**	**Median**	**95th**	**Median**	**95th**
① Seen workload	Linear	**1.25**	2.61	**1.28**	2.99
	2-way-join	**1.36**	3.34	**1.57**	5.68
	3-way-join	**1.34**	3.17	**1.71**	5.97
	Overall	**1.29**	3.09	**1.47**	5.10
② Unseen workload	2-filter-chained	**1.28**	2.61	**1.68**	4.55
	3-filter-chained	**1.26**	2.90	**2.03**	5.68
	4-filter-chained	**1.32**	4.02	**2.08**	6.09
	4-way-join	**1.18**	2.86	**1.59**	4.20
	5-way-join	**1.22**	3.45	**1.77**	4.81
	6-way-join	**1.30**	3.92	**1.81**	6.06
③ Unseen benchmark	Spike Detection	**1.36**	1.68	**2.38**	4.80
	Smart-grid (local)	**1.35**	1.77	**2.47**	3.30
	Smart-grid (global)	**1.32**	1.53	**1.93**	2.64

Table 11: ZeroTune models cost prediction accuracy (Median and 95th percentile) for the naive training strategy, ZT-Random, across seen and unseen PQP using synthetic data and public benchmarks. The models demonstrate high accuracy in predicting costs for both seen and unseen workloads [1, 3].

Few-Shot Learning (FSL): Few-shot learning extends one-shot learning by allowing the model to learn from a few examples rather than just one, requiring significantly fewer data points than traditional machine learning but more than one-shot learning. Following up on previous examples of streaming applications where it needs to predict server failures. With few-shot learning, the model could learn to predict types of failures or performance issues based on only a few instances of each type. For instance, it might learn to predict overloads or hardware malfunctions from just a few labeled examples, facilitating more robust performance monitoring with minimal training data.

Model sees more than one example of test data

A.2.2 *Additional Insight on Performance Prediction*

In this section, we discuss additional results of performance prediction using ZeroTune for ZT-Random training strategy. We show the accuracy and generalization capability ZeroTune for seen and unseen parallel query structures (PQP) as well as varying workload and resource configurations as we showed for OptiSample in Section 6.3.

Accuracy on Seen-Unseen Workload

In this evaluation, we assess the performance of ZeroTune in predicting the cost of seen query structures with a test set (normal train, validation and test set split) unknown to the model but within the (seen) range specified in Table 11. We train and evaluate the ZeroTune models with the enumeration strategies: OptiSample and ZT-Random for each performance metric and three PQP: linear, 2-way join, and 3-way join. We presented the evaluations on OptiSample in Chapter 6: Section 6.3 but not on ZT-Random, which we present in this section. The evaluation focuses on the accuracy of prediction of latency and throughput, using the median and 95th percentile q-error as performance metrics, with 1.0 being a perfect estimate. In addition, we included an "overall" query structure that incorporated data from all three PQP to measure the overall model performance.

Consistent accurate performance for seen data

Seen Workloads: Table 11 ① presents the accuracy of ZeroTune model in predicting latency and throughput for the PQP specifically when ZeroTune is trained with ZT-Random training strategy. Using ZT-Random training strategies, we observe consistent and accurate cost predictions for PQP. We notice that the model performs better for simple linear queries when trained using the ZT-Random strategy. The variance in prediction accuracy may be attributed to the fact that q-errors increase with the complexity of the query. The random enumeration strategy encounters a greater variety of "good" and "bad" query structures, leading to higher variance in q-errors. In general, ZeroTune models for ZT-Random training strategy provide consistently accurate cost predictions for PQP within the training range. In the next step, we will assess the model's performance for PQP outside the training range to understand its generalization capabilities.

Measuring model accuracy on unseen data

Unseen Workloads: An important objective of our study is to evaluate the accuracy and generalization of the ZeroTune model in predicting the cost of unseen PQP that are not encountered during the model's training phase. To achieve this, we increase the complexity of the existing query structures (linear, 2-way join, and 3-way join) by incorporating additional filter and join operators (e.g., 2-filter-chained, 4-way-join), as shown in Table 11②. Subsequently, we generate 200 queries for each template and conduct experiments using ZT-Random strategies to evaluate the accuracy of the *overall* model in predicting the cost for these unseen PQP.

Transfer knowledge from seen to unseen data

ZT-Random strategy yields median latency values ranging from 1.18 to 1.32 for different query structures, with 95^{th} percentile latency values ranging from 2.86 to 4.02. The median throughput values ranged from 1.59 to 2.08, with 95^{th} percentile throughput values ranging from 4.20 to 6.09. These results indicate that the zero-shot model based on the *ZT-Random* strategy can make relatively accurate predictions for the unseen query structures in terms of latency and throughput and generalize to query structures outside the training range.

The results in Table 11② indicate that the model exhibits promising performance in accurately predicting the cost for unseen PQP. While there are some variations in the predicted accuracy, it is noteworthy that the model's accuracy is maintained with the increasing complexity of the unseen PQP. The model demonstrates a level of generalization beyond the training range. The increased complexity is characterized by a higher number of joins and filters, often leading to significantly higher latency and lower throughput. This introduces challenges in accurately predicting the cost due to limited experience with the combinations of parallelism degrees. Nonetheless, such exceptional variations in unseen PQP can be effectively addressed by utilizing the few-shot model as shown in Figure 37 Overall, the results indicate that the zero-shot model exhibits promise in accurately predicting the cost for unseen PQP outside the training range.

Accurate predictions for unseen data

Unseen Benchmarks: In addition to synthetic queries, we conduct further evaluations to assess the accuracy and generalization of our ZeroTune model using three existing benchmark queries: *smart-grid, and spike detection*. The results in Table 11③ illustrate that ZeroTune model demonstrates reasonable accuracy in predicting costs for the benchmark queries. For ZT-Random strategy, we observe variations in latency and throughput across different PQP. These results indicate that both models demonstrate reasonable accuracy in predicting latencies for the benchmark queries.

Accurate prediction for real-world benchmarks

Fine-grained Parallelism Analysis

Next, we drill down into the accuracy of ZeroTune models based on ZT-Random enumeration strategy. To show the accuracy for different parallelism degrees, we divide it into five categories: *XS, S, M, L,* and *XL* in a similar manner as we evaluated OptiSample strategy (cf. Section 6.3.2), which state how much the average parallelism degree is for a query per operator. The following shows a fine-grained analysis of how ZeroTune generalizes for seen and unseen query types and public benchmarks for each parallelism category.

Seen Workloads. In Figure 60, we illustrate the accuracy of the model for seen query types, i.e., for the test set within the range (cf. Table 8). We observe that our ZeroTune model consistently performs well, providing highly accurate cost predictions for different parallelism degrees enumerated based on ZT-Random strategy for seen query types in line with the overall results in Table 11①. From the results, we can observe additional insights into the performance characteristics of PQP, which are influenced by the impact of the complexity of PQP and parallelism categories, as well as the trade-off between latency and throughput. Firstly, an analysis of the influence of increasing PQP complexity on accuracy reveals that more complex PQP generally exhibit marginally lower accuracy compared to simpler PQP. This can be

Influence of parallelism on accuracy

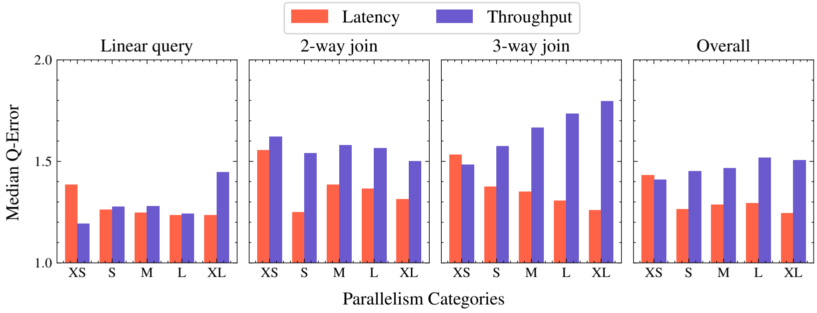


Figure 60: ZeroTune models cost prediction accuracy (Median q-error) of PQP for varying parallelism for seen query structures [1, 3].

attributed to the increased computational overhead and data dependencies associated with complex query structures, making it more challenging for the zero-shot model to predict the performance metrics accurately.

Learns for varying parallelism

High accuracy for seen parallelism

However, it is important to note that the decrease in accuracy is relatively small, indicating that the zero-shot model exhibits reasonable performance in predicting the cost for complex queries. Additionally, the influence of parallelism categories on accuracy is evident. As the parallelism category increases from *XS* to *XL*, we observe a general trend of a decrease in the accuracy of the zero-shot model's predictions for both latency and throughput. This suggests that higher degrees of parallelism provide the ZeroTune model with more information and patterns to learn from, resulting in more difficulty in cost estimation. Moreover, it is worth noting that the decrease in accuracy may not be linear, and there may be diminishing returns as the parallelism category reaches its maximum value. Nevertheless, the differences in accuracy between predictions of different parallelism degrees are relatively small, indicating that the zero-shot model consistently captures the performance characteristics of different queries.

Transfer knowledge to unseen parallelism

Unseen Workload. Next, we assess the accuracy of *ZT*-Random strategy for unseen PQP outside the training range for increasing parallelism categories. Figure 61 presents the results for increased complexity of PQP with added chained filters and joins across different parallelism categories. Significantly, the ZeroTune model consistently demonstrates accurate predictions across various parallelism categories, displaying only minor variations. This indicates that the model successfully identifies performance patterns and generalizes effectively to unseen query plans within this template structure.

However, accuracy slightly decreases for more complex PQP, highlighting the potential challenges in accurately predicting performance metrics for intricate query structures. For instance, for more complex PQP like 5-way and 6-way-join PQP, the ZeroTune model demonstrates a decrease in accuracy, especially for throughput. This is in line with the overall results presented

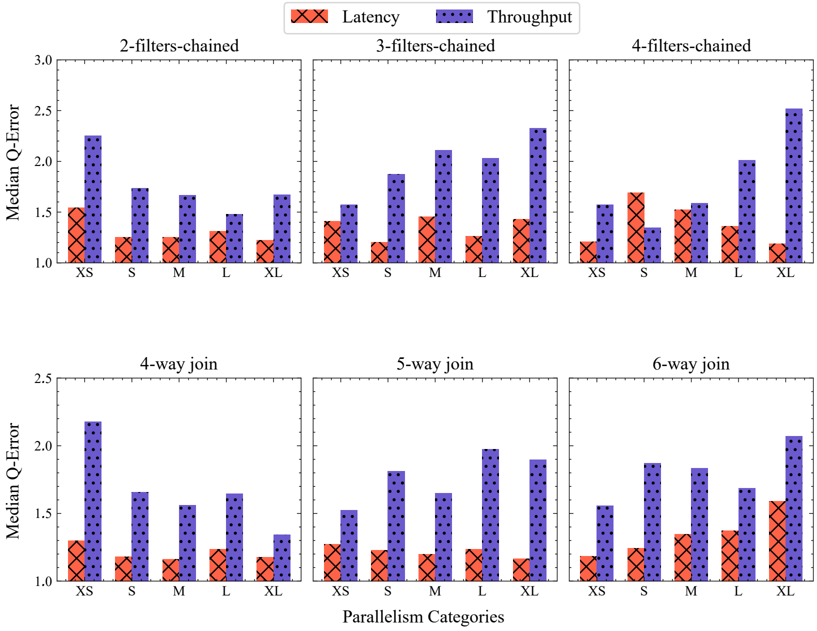


Figure 61: ZeroTune models cost prediction accuracy (Median q-error) of PQP for unseen query structures [1, 3].

in Table 11 ②, which indicated high q-errors for throughput, in particular for 5-way and 6-way joins. However, these variations in performance can easily be enhanced by few-shot learning as we perform for ZeroTune models using OptiSample strategy.

Accurate predictions for unseen parallelism for benchmarks

Unseen Benchmark. We also extend our evaluations to assess the accuracy and generalization capabilities of the ZeroTune model employing the ZT-Random strategy for previously unseen benchmark queries and examine the impact of different parallelism categories. We further evaluate the ZeroTune model using the ZT-Random strategy for unseen benchmark queries and different parallelism categories. The ZT-Random strategy explores a wide range of parallelism categories, from XS to XL, compared to the OptiSample strategy by randomly sampling various workload and query parameters. Results in Figure 40 show that the model accurately predicts costs for different parallelism categories for benchmark queries. However, similar to the OptiSample strategy, prediction errors for throughput are slightly higher than for latency due to differing data distributions between benchmarking and synthetic queries (cf. Table 11 ③). Nevertheless, fine-tuning the model can improve its performance in these cases.

Knowledge transfer to unseen resources

Unseen Resources. In the subsequent step, we assess ZeroTune models based on ZT-Random strategy for their ability to generalize to *unseen* configurations of both heterogeneous and homogeneous resources, as well as the impact of different parallelism categories (cf. to type "U" in Chapter 6: Table 6).

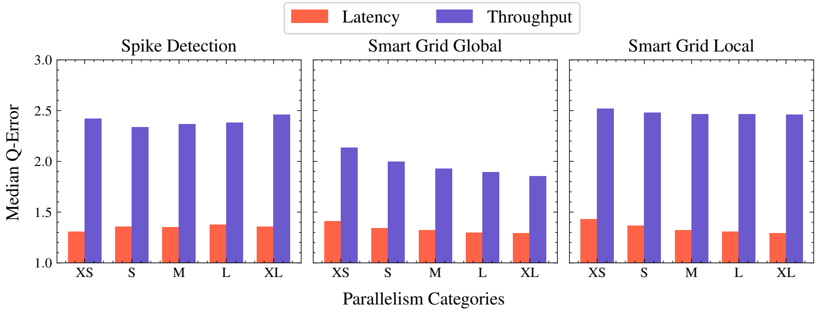


Figure 62: ZeroTune models cost prediction accuracy (Median q-error) of PQP for unseen query benchmark [1, 3].

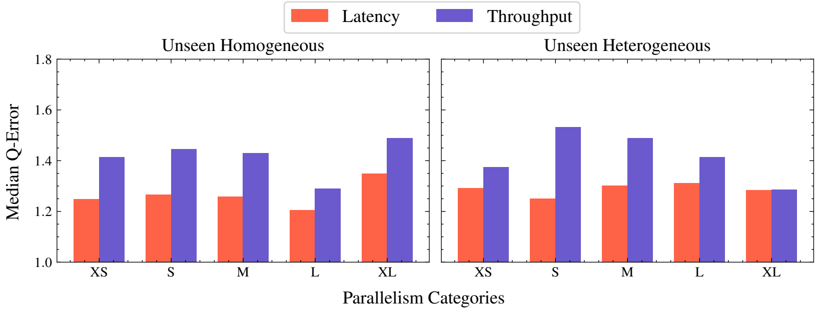


Figure 63: ZeroTune models cost prediction accuracy (Median q-error) of PQP for unseen homogeneous and heterogeneous resources [1, 3].

In Figure 63, ZeroTune model consistently delivers accurate cost predictions also for *unseen* hardware resources for ZT-Random strategy as well.

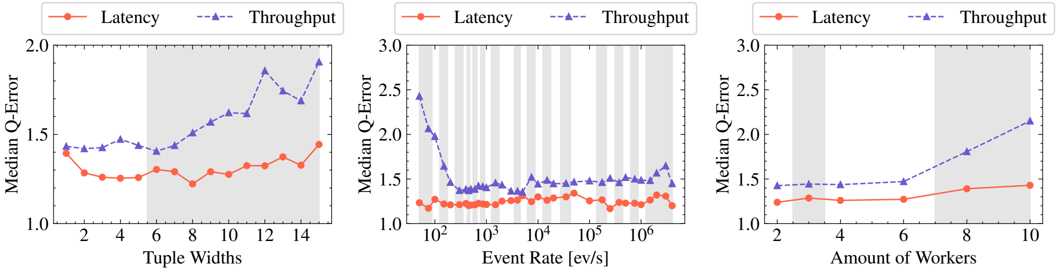
Generalization for Unseen Parameters

Generalization for different workload characteristics

We also evaluate the ZeroTune model's accuracy and generalization with ZT-Random for predicting costs with different unseen workload parameters. By interpolating and extrapolating within the range in Chapter 6: Table 8, we evaluate the performance on unseen parameters like tuple width, window configurations, and available workers in a cluster, as shown in Figures 64 and 65. We use at least 165 queries per tuple width, including a mix of linear, 2-way join, and 3-way join query structures.

High accurate for PQP with different tuple width

Tuple Widths. We show accuracy by extrapolating this parameter because larger tuple widths might benefit from high parallelism degrees. We test on similar query structures with the extrapolation range of $6 - 15$ tuple width. We evaluate the models' performance using at least 165 queries per tuple width, ensuring an equal distribution between linear queries, 2-way join, and 3-way join queries in the PQP. In Figure 64a, we illustrate the performance of ZeroTune model, where we see that the model is very accurate in its



(a) Tuple width (b) Event rate (c) Amount of Workers

Figure 64: ZeroTune models cost prediction accuracy (Median q-error) for PQP with unseen parameters, including (a) tuple widths, (b) event rate, and (c) number of workers. The white section indicates the seen, i.e., within the training range, while the grey section denotes the unseen range of these parameters. ZeroTune generalizes very accurately across all these unseen parameters for ZT-Random strategy [1, 3].

performance prediction and generalizes very well for unseen tuple widths (a grey area). This shows that the model could learn the correlations between the tuple width and the costs of the PQP that generalize when the data stream tuple width increases.

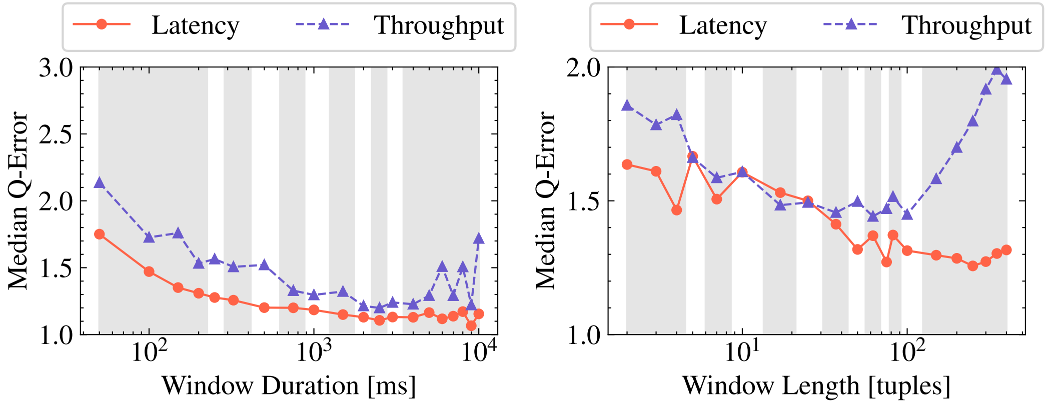
Event Rates. In this evaluation, we assess the capability of the ZeroTune model to inter- and extrapolate event rates beyond the training range (cf Table 8). The model demonstrates high accuracy in predicting costs for both low and high event rates, even outside the training range. The model's ability to predict costs for significantly higher event rates is attributed to its understanding of the system's processing limitations. The model has learned that when the hardware reaches its capacity, resulting in backpressure and increased costs, while Q-error values slightly increase for extremely low event rates. Overall, ZeroTune shows good accuracy and generalization for predicting latency and throughput.

Model learns from high event rates, i.e., backpressure

Amount of workers. The number of workers or nodes influences the performance by affecting available computational resources and parallelism levels. Smaller clusters may limit parallelism, while larger clusters support more parallelism. We evaluate the ZeroTune's accuracy and generalization for seen and unseen number of workers for ZT-Random enumeration strategy. The evaluation results in Figure 64c show that ZeroTune accurately predicts costs for different cluster sizes, enabling the generation of PQP with increasing query complexity and parallelism as cluster sizes grow.

Generalization for varying workers

Window Durations (Time-based). The window duration significantly impacts the cost of PQP by affecting data processing and aggregation of data tuples when the windows are processed in a parallel manner. In this experiment, we include inter- and extrapolation of window duration to demonstrate



(a) Window duration (time-based)

(b) Window length (count-based)

Figure 65: ZeroTune models cost prediction accuracy (Median q-error) of PQP for unseen parameters - (a) window duration (time-based), and (b) window length (count-based). The white area represents the training range, while the grey area shows the unseen range of these parameters. ZeroTune generalizes very accurately across all these unseen parameters also where extrapolation ranges are higher [1, 3].

model generalization for both small and large window sizes. The results in Figure 65a (note log scale on the x-axis) show that ZeroTune accurately predicts costs across various window durations, both seen and unseen training range.

Better understanding of higher window durations

For the shorter window durations, the model starts with a slightly higher median q-error, particularly in the lower range, due to the rapid data processing, making it harder to capture performance characteristics. The model's performance improves as window duration increases, converging to higher accuracy. The Longer window durations allow more time for data accumulation, leading to better understanding and accuracy in latency and throughput predictions. The variations in accuracy are observed towards the extreme ends of the unseen range, likely due to fewer training examples. Overall, ZeroTune generalizes well, especially for longer window sizes, and provides reasonably accurate predictions for shorter window sizes.

ZeroTune learns differently from count and time windows

Window Lengths (Count-based). Much like the window duration, the window length also influences the cost of PQP. However, it is important to note that there is a difference between *time-based* and *count-based* windows in terms of their impact on throughput. When using *time-based* windows, the throughput is generally more constant and independent of the incoming event rate of the operator. While throughput varies with the input rate for count-based windows, predicting throughput for unseen window sizes is more challenging for count-based windows.

Figure 65b shows the accuracy of cost prediction for seen and unseen window lengths. The ZeroTune model also shows good accuracy and general-

Trained Model	**Query**	**Q-error (Latency)**	
ZeroTune	**Structure**	**Median**	**95th**
① No message pre-passing	Validation	**1.3110**	2.3953
	Seen data	**1.2886**	2.3817
	Unseen data	**1.4566**	10.407
	Average	**1.3520**	5.0613
② To↔ Between ↔ On	Validation	**1.2866**	2.6562
	Seen data	**1.2753**	2.9501
	Unseen data	**1.2768**	3.0553
	Average	**1.2795**	2.8872
③ On↔ To ↔ Between	Validation	**1.2589**	2.4812
	Seen data	**1.2612**	2.8484
	Unseen data	**1.2877**	2.7615
	Average	**1.2692**	2.6970
④Between↔ On ↔ To	Validation	**1.2477**	3.9131
	Seen data	**1.2480**	3.8972
	Unseen data	**1.2689**	4.1620
	Average	**1.2548**	3.9907

Table 12: ZeroTune models cost prediction accuracy (Median and 95th percentile) across seen and unseen PQP using variations in order of message passing direction operator **on** resources, operator **to** operator, and **between** resources [1].

ization in predicting costs for different window lengths. However, due to the complexities mentioned earlier, there is a slight increase in q-error for extremely low and high unseen window lengths, particularly for throughput.

A.2.3 *Impact of Message Passing on Prediction Accuracy*

Message passing order between GNN nodes

In the context of the performance prediction model for DSP system, the message passing in Graph Neural Network (GNN) plays a critical role in highly accurate performance prediction to learn from graph representations. For this, we delve into evaluating different orders of message-passing strategies to be utilized by ZeroTune models to optimize latency and throughput predictions under varying workloads. We explore variations in message passing directions—operator **on** resources, operator **to** operator, and **between** resources. This evaluation aims to provide insights into the impact of these message-passing orders on the q-error, a metric used to quantify the accuracy of predictions in terms of relative deviation from actual performance costs.

- No message pre-passing: This baseline configuration does not involve any preliminary message passing between nodes. The results indicate a higher Q-error, especially in unseen data, suggesting limited adaptability to new or dynamic conditions without prior message context.

Impact of different ordering on model's accuracy

- Message passing orders:
  - To↔Between↔On: The message passing begins with operators on resources, moves between resources, and concludes with operator-to-operator communication. It shows a general improvement in q-error across all data types compared to the baseline, indicating effective utilization of resource and operator interdependencies.
  - On↔To↔Between: Altering the sequence to start and end with direct operator interactions (On↔To) with resource-level communications in the middle (Between) yields better performance both in median and tail error for seen and unseen data scenarios, emphasizing the importance of operator-focused communication in familiar contexts.
  - Between↔On↔To: Interactions begin with between resources, followed by the operator on resource, and concluding with operator to operator, resulting in the best average q-error improvement across all tests. This suggests that prioritizing resource-level interactions can enhance model understanding of underlying infrastructure impacts before refining predictions at the operator level.

These results underscore the significance of the order of message passing in enhancing the predictive accuracy of GNN models within zero-shot learning frameworks for DSP systems. The order and focus of message passing not only influence the model's ability to generalize across unseen workloads but also its effectiveness in handling parallelism and resource configuration variations. By tailoring message passing strategies to the specific characteristics of DSP systems, models like ZeroTune can achieve lower q-errors, suggesting more precise predictions of performance costs.

Communication between resources improves accuracy

The results in Table 12, present the median and 95th percentile q-errors for latency cost prediction across different query structures—both seen and unseen. These results underscore the potential of structured message passing in enhancing prediction accuracy. The seen data demonstrates how well the model predicts latency in scenarios similar to those encountered during training, followed by prediction accuracy on unseen data to assess the model's ability to generalize to new, previously not encountered scenarios. At the same time, validation and average scores provide a comprehensive overview of the model's performance across all tested conditions. For results, it is clear that *On↔To↔Between* shows notably balanced performance across diverse seen and unseen workloads, which suggests that prioritizing inter-resource communication potentially enhances the model's ability to capture and generalize the dynamics of distributed stream processing.

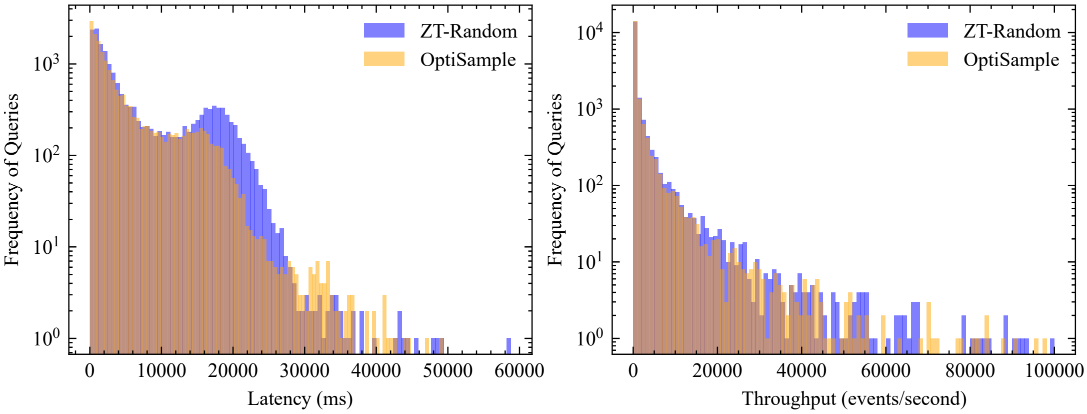


Figure 66: Performance labels (latency and throughput) distribution in training data for overall model with PQP of linear, 2-way and 3-way join PQP [1].

A.2.4 *Distribution of Performance Labels in Training Data*

For training the ZeroTune model to predict performance accurately, it is essential that the model is exposed to a diverse array of workload distributions along with corresponding performance metrics as labels. To facilitate this, we employ a plan generator to create query plans characterized by differing data streams, query parameters, and resource configurations. We utilize both ZT-Random and OptiSample enumeration strategies to explore parallelism within these plans systematically. This approach allows us to comprehensively map the performance metrics—specifically latency and throughput—across the spectrum of generated PQP.

Assessing training data quality

The evaluation aims to assess the effectiveness of the plan generator in providing a broad range of PQP, each with potential variations in performance outcomes, thereby enriching the learning landscape from which the ZeroTune model can derive insights. In Figure 66 presents the distribution of performance metrics, latency, and throughput, comparing the outcomes generated by ZT-Random and OptiSample enumeration strategies across various query plans. The results indicate that these strategies successfully produce a diverse set of PQP, encompassing a wide range of workload and resource characteristics. Crucially, this diversity includes a substantial frequency of different performance ranges, which is fundamental for an effective training and learning process in performance prediction models. This evaluation highlights the capability of our enumeration strategies and plan generator to create a rich dataset, ensuring that the ZeroTune model is well-equipped to learn from varied scenarios and ultimately enhance its predictive accuracy and generalizability.

Effectiveness of model improves with...

...variability of data.

A.2.5 *Demonstration of Performance Modeling*

We provide a demonstration of our proposed performance modeling methods PDSP-Bench and ZeroTune. These YouTube videos show the integration of both PDSP-Bench and ZeroTune in one platform for performance benchmarking and generating corpora of data for training ZeroTune models for accurate performance prediction. The demo is divided into two parts: *(i)* first part presents both solutions *(ii)* second part shows benchmarking processing using PDSP-Bench. For performance benchmarking of learned cost models, the models are already trained for inference through PDSP-Bench.

- Demo part 1: `https://youtu.be/HLQ0moWbIwg`.
- Demo part 2: `https://youtu.be/gs_kTdpwdr4`.

B

Supervised Students

B.1 Master Theses

[1] Chengbo Zhou. "A Collaborative ML-based Network Intrusion Detection Architecture in Software Defined Networking." KOM Best Master Thesis Award: KOM-M-0755. Master Thesis (*co-supervised*). Technical University of Darmstadt, 2023 (cit. on p. xv).

[2] Paul Stiegele. "Learned Parallel Stream Processing using Zero-Shot Cost Models." Nominated for KOM Best Master Thesis Award: KOM-M-0753. Master Thesis. Technical University of Darmstadt, 2023 (cit. on pp. xvi, xvii).

[3] Shubham Sumalya. "Benchmarking System Parallel and Heterogeneous Stream Processing." KOM-M-0770. Master Thesis. Technical University of Darmstadt, 2024 (cit. on p. xvii).

[4] Mizuki Hashimoto. "Optimized Feature Selection Strategy for Zero-Shot Models." KOM-M-0777. Master Thesis. Technical University of Darmstadt, 2023.

[5] Caroline Braams. "Securing Publish Subscribe System using Software-defined Networks." Master Thesis (*co-supervised*). University of Groningen, 2022.

B.2 Labs and Seminars

[1] Chengbo Zhou. "Network Security Monitoring in SDN." ACS-6. Lab Project. Technical University of Darmstadt, 2022 (cit. on p. xv).

[2] Youssef Rafik, Ibrahim Katrin, Ludwig Paula, and Michael Girges. "Parallel and Distributed Streaming Benchmark." Co-supervised with Roman Heinrich and Manisha Luthra. Lab Project. Technical University of Darmstadt, 2024.

[3] Rhivu Mitra. "Auto-Scaling for Distributed Stream Processing using RL." T-11. Lab Project. Technical University of Darmstadt, 2024.

[4] Hamna Naeem. "Efficient Memory Management for Large Language Model Serving with PagedAttention." Seminar. Technical University of Ilmenau, 2024.

[5] Daniel Bondarew. "gSampler: General and Efficient GPU-based Graph Sampling for Graph Learning." Seminar. Technical University of Ilmenau, 2024.

[6] Daniil Ignatev. “Parallel Stream Processing in Distributed Stream Processing Systems.” Seminar. Technical University of Ilmenau, 2023.

[7] Mouayad Mardini. “Parallelizing Intra-Window Join using Hardware Accelerator.” T-13. Seminar. Technical University of Darmstadt, 2023.

[8] Li Zongdi, Chen Jinyao, and Mhd Mouaz Al Jiroudi. “Parallel Stream Processing and Data Analytics using Flink.” T-4. Lab Project. Technical University of Darmstadt, 2023.

[9] Nishchay Dudeja and Chippy Joseph. “Parallel Stream Processing using Reinforcement Learning.” ACS-1. Seminar. Technical University of Darmstadt, 2023.

[10] Mouayad Mardini and Oezcan Karaca. “Real-time query validation and performance visualization.” ACS-2. Lab Project. Technical University of Darmstadt, 2023.

[11] Harshala Bhavesh, Sravya Neelam, and Gayatri Marreddy. “Big Data Analysis and Performance Evaluation.” T-1. Lab Project. Technical University of Darmstadt, 2023.

[12] Aastha Bajirao and Sushmitha Chandirasegaran. “Learned Operator Parallelization and Stream Processing.” ACS-7. Seminar. Technical University of Darmstadt, 2022.

[13] Christian Schwenk and Sushmitha chandirasegaran. “Query processing and performance analysis using Flink.” ACS-12. Lab Project. Technical University of Darmstadt, 2022.

[14] Anton Winter. “Performance Evaluation of Windows Join in Stream Processing System.” ACS-15. Lab Project. Technical University of Darmstadt, 2022.

[15] Marvin Joos. “Hardware Accelerated Stream Processing.” ACS-9. Seminar. Technical University of Darmstadt, 2021.

[16] Ali Acheche. “Resources Elasticity and Computation Offloading Techniques in Stream Processing Systems.” ACS-8. Seminar. Technical University of Darmstadt, 2021.

[17] Corinna Augustin. “Telcaria Alviu: Network Traffic Monitoring and Profiling.” ACS-11. Lab Project. Technical University of Darmstadt, 2021.

C

Erklärung laut Promotionsordnung

§ 8 Abs. 1 lit. d PromO

Ich versichere hiermit, dass von mir zu keinem vorherigen Zeitpunkt bereits ein Promotionsversuch unternommen wurde. Andernfalls versichere ich, dass der promotionsführende Fachbereich über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs informiert ist.

§ 9 Abs. 1 PromO

Ich versichere hiermit, dass die vorliegende Dissertation, abgesehen von den in ihr ausdrücklich genannten Hilfsmitteln, selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde. Weiterhin versichere ich, dass die "Grundsätze zur Sicherung guter wissenschaftlicher Praxis an der Technischen Universität Darmstadt" sowie die Leitlinien zum Umgang mit digitalen Forschungsdaten an der TU Darmstadt" in den jeweils aktuellen Versionen bei der Verfassung der Dissertation beachtet wurden.

§ 9 Abs. 2 PromO

Ich versichere hiermit, dass die vorliegende Dissertation bisher noch nicht zu Prüfungszwecken gedient hat.

Darmstadt, June 25, 2024

Pratyush Agnihotri

Colophon

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

`https://bitbucket.org/amiede/classicthesis/`

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

`http://postcards.miede.de/`

Final Version as of October 10, 2024 (`classicthesis`).