Graph representation learning for predictive quality in multi-stage discrete manufacturing

Graphenrepräsentationslernen für Qualitätsprädiktion in mehrstufigen diskreten Produktionsprozessen

Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
Genehmigte Dissertation von Beatriz Bretones Cassoli Nevejan aus Santa Rita do Passa Quatro, SP, Brasilien
Tag der Einreichung: 19.03.2024, Tag der Prüfung: 02.07.2024

1. Gutachten: Prof. Dr.-Ing. Joachim Metternich
2. Gutachten: Prof. Dr.-Ing. Dipl.-Wirtsch.-Ing. Peter Groche

TECHNISCHE UNIVERSITÄT DARMSTADT

Mechanical Engineering Department

Institute for Production Management, Technology and Machine Tools

Management of Industrial Production

Graph representation learning for predictive quality in multi-stage discrete manufacturing
Graphenrepräsentationslernen für Qualitätsprädiktion in mehrstufigen diskreten Produktionsprozessen

Accepted doctoral thesis by Beatriz Bretones Cassoli Nevejan

Date of submission: 19.03.2024
Date of thesis defense: 02.07.2024

Vorwort des Herausgebers

Produzierende Unternehmen stehen vor der Herausforderung, ihre Fertigungsprozesse kontinuierlich zu verbessern. Die gesteigerte Verfügbarkeit von Daten verspricht u.a. auf dem Gebiet der Qualitätssicherung die Möglichkeit, die Qualität von Teilen und Produkten vorherzusagen, ohne diese messen oder testen zu müssen. Mit der vorliegenden Arbeit greift Frau Bretones Cassoli Nevejan diese Themenstellung im Kontext mehrstufiger diskreter Fertigungsprozesse auf.

Ein zentrales Merkmal der Arbeit ist der innovative Einsatz von Graph Neural Networks (GNNs) zur Modellierung und Analyse von Fertigungsprozessen. Mithilfe der Graphenrepräsentationen werden die komplexen Abhängigkeiten und Beziehungen innerhalb von Produktionsabläufen abgebildet. Durch die Darstellung von Fertigungsprozessen als Graphen können die Wechselwirkungen zwischen verschiedenen Prozessschritten und Komponenten erfasst und für die Qualitätsprädiktion nutzbar gemacht werden.

GNNs bieten eine leistungsstarke Methode zur Verarbeitung und Analyse solcher graphbasierter Datenstrukturen. Diese Netzwerke sind speziell darauf ausgelegt, die Verbindungen und Zusammenhänge innerhalb von Graphen zu lernen und zu interpretieren. Der Einsatz von GNNs ermöglicht es, die vielschichtigen Beziehungen und zeitlichen Abhängigkeiten in der Produktion zu modellieren und erlaubt somit eine umfassendere und präzisere Klassifikation der Produktqualität als andere Machine Learning Ansätze.

Die Dissertation zeigt durch umfassende theoretische und empirische Untersuchungen die Überlegenheit dieses graphbasierten Modells im Vergleich zu konventionellen maschinellen Lernverfahren. Die vorgestellten Fallstudien zeigen im Einzelfall eine klare Verbesserung in der Klassifikation der Produktqualität und unterstreichen das Potenzial dieses Ansatzes zur Rationalisierung der Qualitätskontrolle in der industriellen Praxis.

In Summe leistet die Arbeit damit einen wertvollen Beitrag zur Weiterentwicklung der prädiktiven Qualitätskontrolle und stellt einen wichtigen Schritt zur Verbesserung der betrieblichen Effizienz und Produktqualität in der modernen Fertigung dar.

Darmstadt, Juli 2024 Prof. Dr.-Ing. Joachim Metternich

Acknowledgments

I want to express my gratitude to my supervisor, Professor Metternich, for his guidance and productive discussions and for helping me maintain a clear focus on the manufacturing problem while defining my research topic. I am also thankful for the opportunity to have worked as a research associate at the Institute for Production Management, Technology, and Machine Tools (PTW) at the Technical University of Darmstadt (TUDa).

My academic journey at the TUDa would not have been possible without the opportunities the Escola Politécnica of the University of São Paulo (Poli) provided. I thank Professors Paulo Carlos Kaminski and Eduardo Zancul, my Poli supervisors, and Professor Marcio Lobo Neto for coordinating and enabling the double degree program with the TUDa. I am also grateful to Professor Fabiano Armellini from Polytechnique Montreal, whose support was instrumental in my decision to pursue a doctorate.

I thank all my professors and teachers for their dedication and support throughout my educational journey.

I want to thank my colleague and friend, Nicolas Jourdan, for his feedback on my research, help with machine learning topics, and manuscript revision, which significantly shaped this dissertation. I am also thankful to have been part of the Management of Industrial Production (MiP) group. I thank my colleague and friend Felix Hoffmann for helping me navigate the challenges of working in a German-speaking institute and Nik Weisbrod for revising my thesis. I also thank my former colleague Amina Ziegenbein for hiring me as a student assistant and referring me to the research associate position. While I cannot name all my colleagues, I sincerely appreciate the friendly work environment provided by everyone in the MiP group and at PTW. I would also like to acknowledge the students who assisted in recording the dataset at the CiP learning factory, which I used in my experiments.

On a personal note, I want to express my gratitude to my family and friends. Dear Mom Ana Cristina, your constant support, love, and dedication have been the foundation of my academic success. Your sacrifices and determination have inspired me every step of the way. Tia Imma and tio Constante, thank you for your lifelong love and support. Tio, your guidance and belief in me have continuously inspired me. I am grateful for your help revising my thesis and offering advice whenever needed. While I cannot mention everyone by name, please know that I cherish and appreciate all the encouragement and support I have received from family and friends throughout this journey.

My beloved husband, Bruno, thank you; your encouragement, patience, and love have been my anchor throughout this process. Your support and continuous motivation during the challenging moments of my doctorate were indispensable. I could not have achieved this without you, and I am deeply grateful to have you by my side.

Finally, I express my deepest gratitude to God. I draw inspiration from the words of Professor Gombolay, which I fully share: 'As a research community, every challenge we face and discovery we make instills in me even more appreciation for the LORD's infinite knowledge, which created the heavens and earth.' Throughout my doctoral journey, my faith in Him has been my source of strength, and I have learned to trust in Him and embrace the belief that 'I can do all things through Christ who strengthens me.'

Darmstadt, July 2024 Beatriz Bretones Cassoli Nevejan

Abstract

This thesis addresses the need for predictive quality control methods in multi-stage discrete manufacturing processes to enhance operational efficiency and product quality. Traditional end-of-line quality control practices are being replaced or supplemented by data-driven approaches to mitigate manual testing dependencies. Leveraging graph representation learning, inspired by successful applications in biochemistry, this research proposes a novel approach for predictive quality control, aiming to capture complex interdependencies within manufacturing processes. The primary objective is to evaluate the efficacy of this approach compared to alternative machine learning methods.

Through comprehensive theoretical and empirical analysis, this thesis makes two significant contributions. Firstly, it introduces a novel graph representation learning approach tailored for manufacturing data modeling and product quality classification. This approach offers a systematic guideline for implementation across diverse manufacturing contexts. Secondly, empirical validation through two distinct case studies demonstrates the superior performance of the proposed method over conventional machine learning techniques. The results support its potential for enhancing product quality classification and streamlining quality control operations in multi-stage discrete manufacturing.

Overall, this research contributes to advancing predictive quality control methodologies in multi-stage discrete manufacturing processes, offering practical insights and guidelines for industry adoption in pursuit of enhanced operational efficiency and product quality.

Zusammenfassung

Diese Dissertation befasst sich mit dem Bedarf an Methoden für prädiktive Qualitätskontrolle in mehrstufigen diskreten Fertigungsprozessen zur Verbesserung der betrieblichen Effizienz und der Produktqualität. Bestehende Verfahren zur Qualitätskontrolle am Ende der Fertigungslinie werden durch datengesteuerte Ansätze ersetzt oder ergänzt, um die Abhängigkeit von manuellen Prüfungen zu vermindern. Unter Nutzung des Lernens von Graphenrepräsentationen, inspiriert durch erfolgreiche Anwendungen in der Biochemie, wird in dieser Forschungsarbeit ein neuartiger Ansatz für die prädiktive Qualitätskontrolle vorgeschlagen, der darauf abzielt, komplexe Abhängigkeiten innerhalb von Fertigungsprozessen zu erfassen. Das Hauptziel ist die Bewertung der Effektivität dieses Ansatzes im Vergleich zu alternativen maschinellen Lernmethoden.

Durch eine umfassende theoretische und empirische Analyse leistet diese Dissertation zwei wichtige Beiträge. Erstens wird ein neuartiger Lernansatz auf Basis von Graphenrepräsentationen vorgestellt, der auf die Modellierung von Produktionsdaten und die Klassifizierung der Produktqualität zugeschnitten ist. Dieser Ansatz bietet einen systematischen Leitfaden für die Implementierung in unterschiedlichen Fertigungskontexten. Zweitens zeigt die empirische Validierung anhand von zwei verschiedenen Fallstudien eine Steigerung der Klassifikationsleistung der vorgeschlagenen Methode gegenüber konventionellen maschinellen Lernverfahren. Die Ergebnisse belegen das Potenzial der Methode zur Verbesserung der Produktqualitätsklassifizierung und zur Rationalisierung von Qualitätskontrollverfahren in der mehrstufigen diskreten Fertigung.

Insgesamt trägt diese Forschungsarbeit dazu bei, die Anwendung prädiktiver Qualitätskontrollmethoden in mehrstufigen diskreten Fertigungsprozessen voranzutreiben. Sie präsentiert zudem praktische Einblicke und Richtlinien für die Anwendung in der Industrie, um die betriebliche Effizienz und Produktqualität zu verbessern.

Contents

List of figures

List of tables

List of acronyms

ACC Accuracy. 38

AUC-ROC Area Under the Receiver Operator Characteristic Curve. 38, 39, 82, 83, 85, 97, 99–101

BCEL Binary Cross-Entropy Loss. 19

CiP Center for Industrial Productivity. 66, 86, 90, 91

CiP-DMD CiP Discrete Manufacturing Dataset. xv, 66, 86, 89–91, 105–107, 109, 110, 115

CNN Convolutional Neural Network. 45, 58, 61

CRISP-ML(Q) CRoss-Industry Standard Process model for the development of Machine Learning applications with Quality assurance methodology. xiii, 65, 110

CSV Comma Separated Values. 73

CWT Continuous Wavelet Transform. 25

DFT Discrete Fourier Transform. 24, 25

DGL Deep Graph Library. 69, 70

DWT Discrete Wavelet Transform. 25

ERR Prediction Error. 38

FFT Fast Fourier Transform. 25

FMEA Failure Mode and Effects Analysis. 12

FPR False Positive Rate. 38

GCN Graph Convolutional Network. 45

GIN Graph Isomorphism Network Layer. 47–49, 81, 96

GNN Graph Neural Network. xiii, xv, 5, 41, 44–47, 62, 64, 67, 69–71, 73, 77, 78, 80, 82, 83, 85, 90, 92, 93, 95–107, 109–111

GraphConv Graph Convolutional Layer. 47, 81, 96

GraphSAGE GraphSAGE Layer. 47, 48, 81, 96, 107

i.i.d. independent and identically distributed. 44

IoT Internet of Things. 14, 52

ISO International Organization for Standardization. 13

KG Knowledge Graph. xiii, 42–44, 61

kNN k-Nearest Neighbor. 29, 32, 33, 58, 61, 64, 70, 71, 81, 95, 100, 110

LR Logistic Regression. 19, 29, 30, 61, 64, 70, 71, 81, 83, 95, 101, 110

LSTM Long Short-Term Memory. 61

MCC Matthews Correlation Coefficient. 40, 41, 82–84, 97, 99–101, 103

ML Machine Learning. xiii, 1, 3, 5, 6, 14–18, 20–22, 27, 38, 41, 44, 47, 57–65, 67, 70, 73, 107, 109–111

MLP Multilayer Perceptron. 29, 34–36, 61, 64, 70, 71, 81, 96, 98, 110

MPS Minimal Processing Steps. 67, 68, 76, 90, 106

MQTT Message Queuing Telemetry Transport. 87

NC Numerical Control. 87

OEE Overall Equipment Effectiveness. 9, 10

OPC UA Open Platform Communications Unified Architecture. 87

PCB Printed Circuit Board. 59, 62

PSD Power Spectral Density. 26

PTW Institute for Production Management, Technology and Machine Tools. 86

RF Random Forest. 29–31, 33, 58, 61, 64, 70, 71, 81, 83, 95, 97, 110

ROC Receiver Operator Characteristic. xiii, 38, 39

RQ Research Question. 53, 59

SGD Stochastic Gradient Descent. 18, 19

SPC Statistical Process Control. 11

SVM Support Vector Machine. xxi, 19, 29, 31, 32, 58, 61, 64, 70, 71, 81, 95, 110

TPR True Positive Rate. 38, 39

TPS Toyota Production System. 12, 14

TQM Total Quality Management. 12

tsfresh Time Series FeatuRe Extraction based on Scalable Hypothesis tests. xv, 68, 77, 92, 115

TUDa Technical University of Darmstadt. 86

XGBoost eXtreme Gradient Boosting. xxii, 33, 34, 61, 64, 70, 71, 81, 83, 95, 98, 100, 101, 104, 110

List of symbols

Symbol	Description
n	Total number of instances in the dataset
y_i	Actual binary label (0 or 1) of the i-th instance
p_i	Predicted probability that the i-th instance belongs to class 1
$f(x_i)$	Raw output of the decision function
$f(x)$	Raw output of the logistic regression model
$\bigtriangledown_{\mathbf{x}} f(\mathbf{x})$	Gradient of the function with respect to input variables
$\mathbf{x}$	Input variables of a function
θ	Model parameters
ϵ	Learning rate
J	Regularized objective function
λ	Strength of regularization
$\bar{x}$	Arithmetic mean or average
σ	Standard deviation
σ^2	Variance
$X(f)$	Frequency representation of the signal in the frequency domain
j	Imaginary unit
$x(t)$	Signal in the time domain
$x[n]$	Discrete signal in the time domain
$X[k]$	Frequency representation of the signal at frequency index k
N	Length of the signal
$\psi(t)$	Analyzing wavelet
a	Scale parameter
b	Translation parameter
ψ^*	Complex conjugate of the analyzing wavelet
$h[n]$	Low-pass filter
$g[n]$	High-pass filter
$A_{o,k}$	Approximation coefficients at scale o and position k
$D_{o,k}$	Detail coefficients at scale o and position k
U	Mann-Whitney U statistic
Φ	Cumulative distribution function of the standard normal distribution
m	Total number of features
p_0	Proportion of instances belonging to class 0
p_1	Proportion of instances belonging to class 1
ξ_i	Slack variable
$\phi(\mathbf{x_i})$	Kernel function
C	Regularization parameter SVM classifier
$d(\mathbf{x_i}, \mathbf{x_j})$	Distance between two data items $\mathbf{x_i}$ and $\mathbf{x_j}$

Symbol	Description
$\ell(y_i, \hat{y}_i)$	Logistic loss
K	Number of weak learners (trees) XGBoost
$\Omega(f_k)$	Regularization term XGBoost
T	Number of leaves in the tree XGBoost
G	Graph
V	Node set
E	Edge set
$\mathbf{A}$	Adjacency matrix
$\eta \colon V \to S$	Node type mapping function
$\varphi \colon E \to R$	Edge type mapping function
S	Node types
R	Edge types
$m_{N(u)}$	Message
h_u	Embedding
$N(u)$	Nodes u's graph neighborhood
$\mathbf{H}^{(k)}$	Matrix of node representations at layer k
$\mathbf{W}_{neigh}^{(k)}$	Weight matrix message passing operation from neighboring nodes at layer k
$\mathbf{W}_{self}^{(k)}$	Weight matrix self-connections of nodes at layer k
$\mathbf{B}^{(k)}$	Bias term for layer k
π	Element-wise nonlinearity (activation function)
c_{ji}	Product of the square root of node degrees

1. Introduction

Discrete manufacturing processes produce distinct, individual products that can be counted and distinguished as separate items, contrasting with continuous manufacturing processes, which continuously produce goods in a steady stream [34]. A multi-stage discrete manufacturing process is characterized by $n > 1$ intermediate steps, often called stations, workstations or simply sub-processes, where raw materials or components are progressively transformed into finished products [98]. These systems incorporate various layouts, including setups with dedicated machines, flexible machinery, assembly lines, disassembly lines, and serial production lines [26].

In a multi-stage process, each manufacturing step yields output that is the input for a subsequent step. This interdependence creates complex relationships between process variables and the ultimate product's quality. Amid the complexities introduced by these distinct relationship types, suboptimal manufacturing situations such as poor quality of raw materials, machinery malfunction, or breakdowns, coupled with the inherently stochastic nature of production stages, contribute to deviations from intended design specifications and tolerances specified in a tolerance chain [88]. These deviations manifest as variations in product quality, which can accumulate and propagate through processes [38]. Despite implementing process-specific, in-line quality controls throughout production, a final product quality inspection is often indispensable, especially in situations in which quality conform components result in a non-quality conform final product, as depicted in Figure 1.1. This figure illustrates the relationship between the quality of parts and processes and the final product quality: Non-quality conform parts and processes (represented by the category 'NOK') lead to the production of non-quality conform final products. However, quality-conforming ('OK') parts and processes can result in both OK and NOK final products. This need for a final quality inspection arises from the accumulation of variations at different stages of the manufacturing process and the need to comply with legal requirements and verify adherence to industry regulations and standards [42]. Additionally, final product quality control is crucial in risk minimization to identify and mitigate potential issues before the product reaches the market [42].

The complex nature of multi-stage discrete manufacturing systems requires dedicated consideration and a comprehensive approach when assessing and automating product quality control. In recent years, there has been a noteworthy surge in the adoption of data-driven approaches for quality control within complex manufacturing systems [123]. Leveraging advancements in Machine Learning (ML), statistical analysis, and data analytics, researchers and industry practitioners have been exploring innovative ways to harness the vast amounts of process data generated during manufacturing. These data-driven methods hold promise in identifying patterns and correlations within complex relationships and predicting and pre-emptively addressing potential deviations from desired quality outcomes. Particularly for multi-stage processes, handling multiple data sources and selecting meaningful features for quality prediction are ongoing challenges to be addressed [103].

Smart manufacturing, also known as intelligent manufacturing, integrates digital systems capable of adapting to dynamic environments and evolving customer demands. Moreover, it encompasses the utilization of ML technologies to enhance the synergy between human operators and automated systems [109]. In 2022, the smart manufacturing market presented an estimated worth of $186.68 million, projected to experience a robust compound annual growth rate of 6.1%. By 2030, this trajectory is forecasted to elevate its value to

approximately $301.5 million. Notably, a comprehensive study revealed that the automotive, semiconductor, and mining sectors have emerged as the leaders in embracing smart manufacturing technologies [127]. A recent analysis of the Chinese manufacturing market has underscored a positive moderating effect on the relationship between intelligent manufacturing adoption and labor productivity. This finding suggests that firms operating within highly competitive markets stand to gain significant productivity enhancements by implementing smart manufacturing initiatives. The need for real-time, data-driven decision-making is particularly pronounced in industries characterized by high competition. Smart manufacturing, leveraging data analytics and process optimization, presents a strategic edge aligning with the fast-paced dynamics of competitive markets [127].

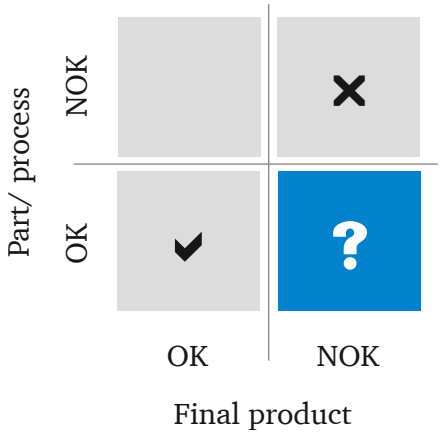


Figure 1.1.: Relationship between part and process quality and final product quality

Predictive quality applications constitute a vital component of smart manufacturing initiatives, using manufacturing data to assess product quality proactively. These initiatives foster a more sustainable production landscape by minimizing the production of defective items and scrap parts, simultaneously reducing costs and saving time [39]. Furthermore, the automation potential inherent in predictive quality initiatives offers the prospect of streamlining quality control operations, thereby mitigating the dependence on manual intervention. This is particularly relevant in regions such as Germany, where a shortage of skilled workers poses a significant challenge to businesses. Indeed, many enterprises view the scarcity of qualified personnel as the number one risk to future operations, recognizing the potential for operational disruptions in the absence of suitable skilled labor [100].

By embracing smart manufacturing technologies, particularly predictive quality initiatives, to address these challenges, businesses can strengthen their competitive position, boost operational efficiency, and adeptly navigate the evolving landscape of modern manufacturing, addressing both economic and ecological concerns.

1.1. General objective and thesis outline

General objective This thesis addresses identified gaps for implementing predictive quality in multi-stage discrete manufacturing processes, aiming to replace or reduce the reliance on manual or physical testing end-of-line quality control. The proposed approach suggests leveraging graph representation learning to improve quality control methods. The primary objective is thus to evaluate the use of this technology for predictive quality applications in manufacturing. The thesis hypothesis asserts that representing manufacturing data as a graph enhances data expressiveness, thus improving the learning process and resulting in superior performance in classification metrics.

The inspiration for adopting a graph-based approach stems from biochemistry, particularly the successful classification of molecules [56]. Graph representation learning, proven effective in capturing complex relationships within molecular structures, can potentially capture interdependencies in multi-stage discrete manufacturing processes. Drawing parallels between molecular structures, characterized by interconnected atoms, and manufacturing processes, which feature interconnected workstations, this work seeks to replicate the success observed in another research field, thereby enhancing quality control and contributing to more streamlined and cost-effective manufacturing processes. More specifically, this thesis introduces an innovative approach to product quality classification. Through rigorous testing and evaluation in two distinct case studies, the superiority of this approach over alternative ML methods is demonstrated.

This thesis contributes to quality control in multi-stage discrete manufacturing processes, addressing critical challenges and proposing innovative solutions. Its contributions can be summarized as follows:

1. **Proposal of a graph representation learning approach:** The primary contribution is the introduction of a novel approach for data modeling and product quality classification leveraging graph representation learning. The proposed approach aims to minimize or replace end-of-line quality control with predictive techniques, arguing that representing manufacturing data as a graph enhances the learning process. The proposed approach is defined by a comprehensive guideline detailing how to implement it for other datasets, facilitating its adoption in diverse manufacturing contexts.

2. **Demonstration of the approach's superiority through case studies:** Through experiments conducted in two case studies, the thesis provides empirical evidence of the superiority of the proposed graph representation learning method. The results validate the approach's effectiveness and demonstrates its practical applicability and potential for outperforming alternative ML methods. The empirical results underscore the promise of this innovative approach for product quality classification.

These contributions collectively advance the understanding and implementation of a novel technology for predictive quality in multi-stage discrete manufacturing processes.

Thesis outline Figure 1.2 illustrates the structure of the thesis. Chapter 2 provides the theoretical foundations of topics relevant to this thesis: multi-stage discrete manufacturing, quality management, ML, and graph representation learning. Chapter 3 presents a comprehensive literature review on predictive quality in multi-stage discrete manufacturing processes, outlining existing approaches and utilized ML models. The chapter identifies gaps in existing research, thereby establishing the context for the contributions made by this thesis. Chapter 4 articulates the research design, including the research question and hypothesis that guide the empirical investigation. The primary contributions of this thesis are elaborated in Chapters 5 and 6, highlighted in green in Figure 1.2. The proposed approach is detailed in Chapter 5. Subsequently, Chapter 6 validates the proposed approach through two case studies. The chapter concludes with a thorough discussion of the empirical findings and practical recommendations for implementing the proposed approach in similar contexts. Finally, Chapter 7 addresses the research question, evaluates the proposed hypothesis, and offers a succinct summary of the thesis, along with its limitations and avenues for potential future research.

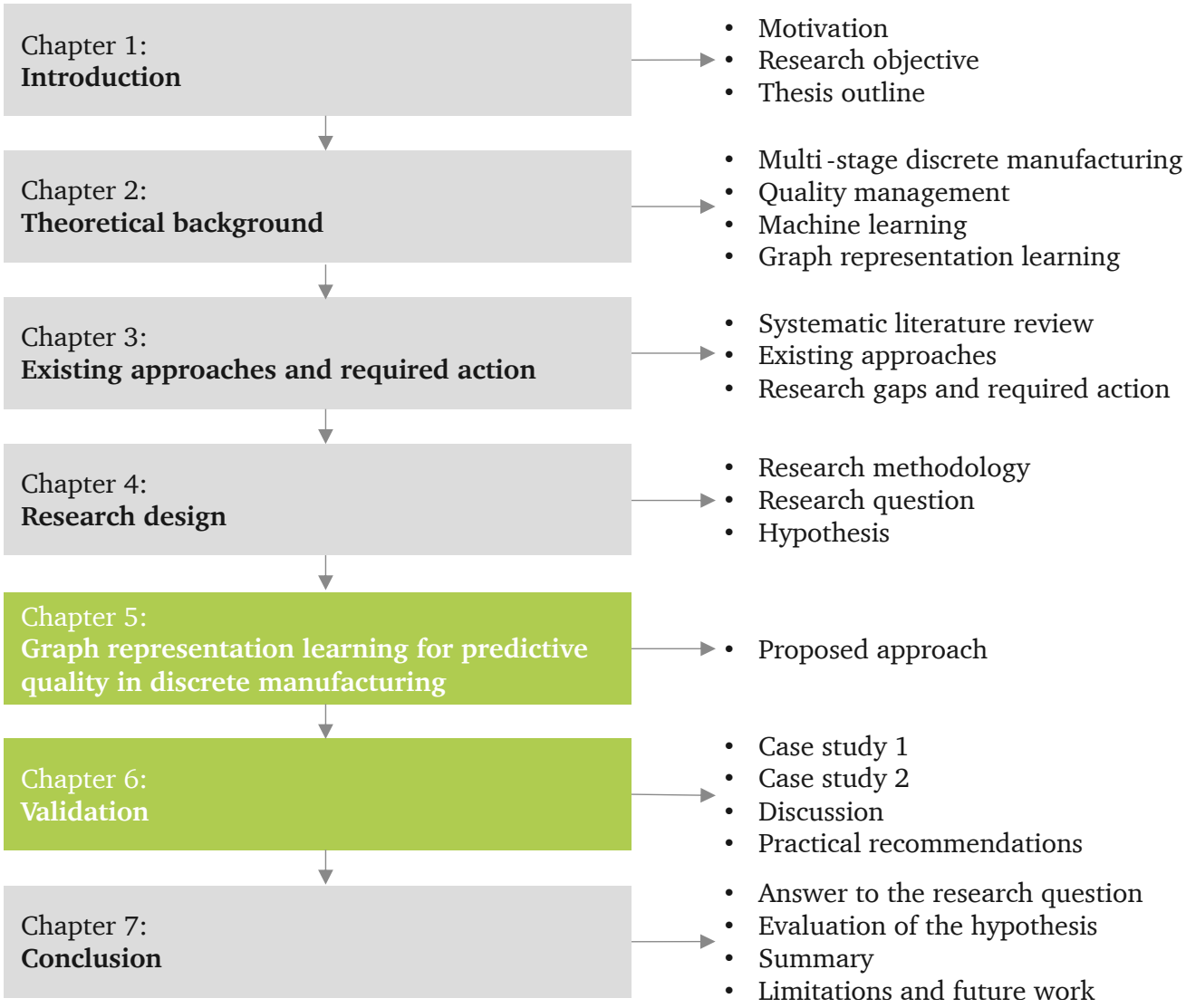


Figure 1.2.: Thesis structure. Main contributions highlighted in green

2. Theoretical background

This chapter lays the foundations for understanding the problem addressed in this thesis. It begins with defining multi-stage discrete manufacturing in Section 2.1, outlining the specific manufacturing processes under consideration. Moving to Section 2.2, the context of quality management is presented, explaining the terms 'quality' and 'quality management' and detailing the four essential tasks within this domain, namely quality planning, assurance, control and improvement. The section presents three quality perspectives within a manufacturing company—product, process, and system quality, emphasizing numerical indicators for quality assessment in each context. Additionally, a brief historical overview and recent developments in quality management connect with the central theme of this work: predictive quality for multi-stage discrete manufacturing processes.

In Section 2.3, an introduction to ML is provided, elucidating key concepts central to the proposed approach aimed at enhancing quality control. This section focuses on classification tasks, introducing ML algorithms used as benchmarks to evaluate the proposed solution.

To conclude the theoretical background chapter, Section 2.4 introduces graph theory and Graph Neural Network (GNN), the technologies explored and assessed in this work for representing the multi-stage manufacturing process and executing the binary classification task.

2.1. Multi-stage discrete manufacturing

Discrete manufacturing is characterized by single part or product production in an individual processing mode, by transformation of raw materials [34]. A multi-stage manufacturing process is characterized by a sequence of $n > 1$ intermediate steps, often referred to as stations or workstations, where raw materials or components are progressively transformed into finished products [98]. As previously mentioned, these multi-stage manufacturing systems encompass diverse layouts, integrating dedicated machines, flexible machinery, assembly, disassembly, and serial production lines [26]. Examples of discrete manufacturing processes include automotive manufacturing, involving the assembly of individual components to produce cars; electronics manufacturing, encompassing the production of printed circuit boards and assembly of components to create electronic appliances; aerospace manufacturing, entailing the production and assembly of aircraft components; and medical device manufacturing, dedicated to producing surgical instruments and diagnostic devices [48].

In a multi-stage process, each manufacturing step yields output that serves as input for the subsequent step. This interdependence gives rise to complex relationships between process variables and the ultimate quality of the end product. Arif et al. [3] categorize the types of these relationships:

- Intra-workstation relationships: Process variables within a workstation are interconnected and collectively influence the output quality of that specific station.
- Inter-workstation relationships: The quality of output from a given workstation is not solely shaped by the processes within that station; it is also influenced by the preceding workstation's output.
- Global process-product relationships: The overall quality of the final product is a culmination of the contributions of all process variables across the entire manufacturing process.

Amid the complexities introduced by these distinct relationship types, suboptimal manufacturing situations such as poor quality of raw materials, machinery malfunction, or breakdowns, coupled with the inherently stochastic nature of production stages, contribute to deviations from intended design specifications [88]. These deviations manifest as variations in product quality, which can accumulate and propagate through multi-stage manufacturing processes [38]. Therefore, the complex nature of multi-stage discrete manufacturing systems necessitates dedicated consideration and a comprehensive approach when assessing and ensuring product quality.

In recent years, there has been a noteworthy surge in the adoption of data-driven approaches for quality control within complex manufacturing systems [123]. Leveraging advancements in ML, statistical analysis, and data analytics, researchers and industry practitioners have been exploring innovative ways to harness the vast amounts of process data generated during multi-stage manufacturing. These data-driven methodologies hold promise in identifying patterns and correlations within complex relationships and predicting and preemptively addressing potential deviations from desired quality outcomes. Section 2.2 introduces quality management, specifically focusing on predictive quality and the use of ML for quality classification in multi-stage processes.

2.2. Quality management

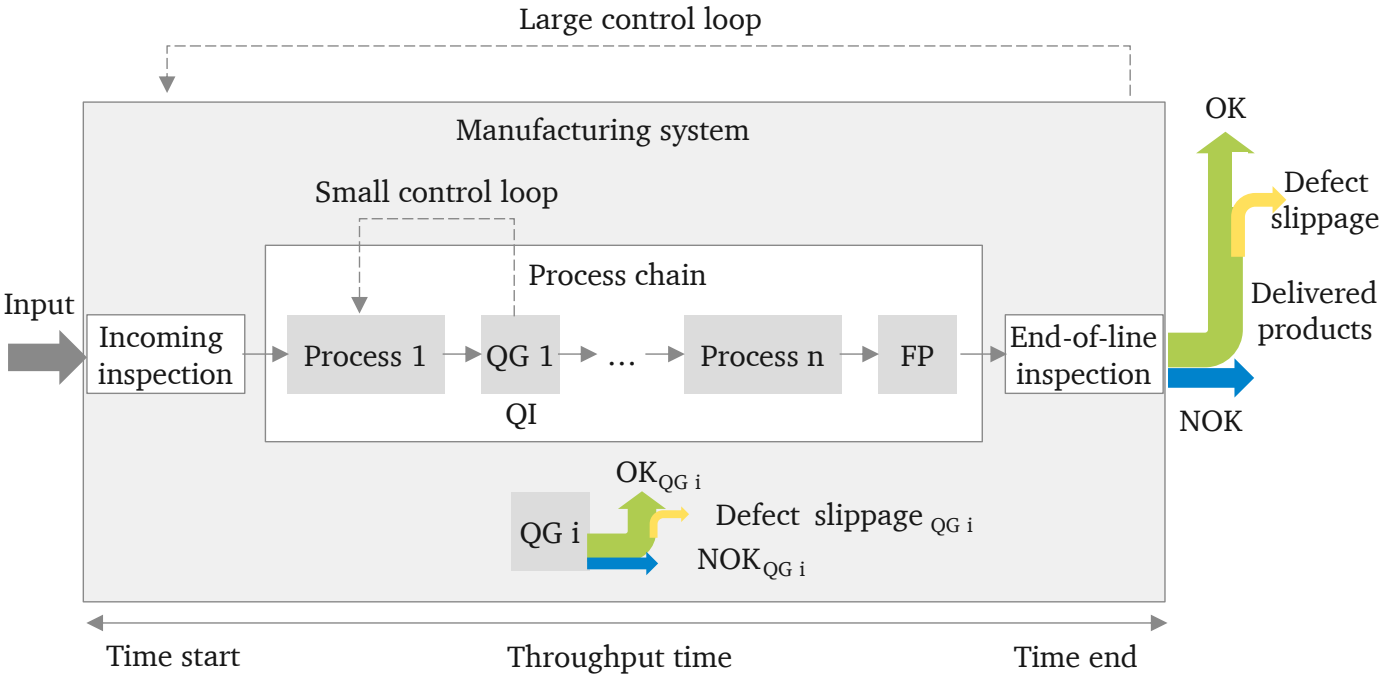
In manufacturing, 'quality' is defined as 'the degree to which a set of inherent characteristics of an object fulfills requirements', DIN EN ISO 9001 [29]. This definition emphasizes the critical importance of meeting specified criteria to achieve a desired level of product excellence. Quality management, as described by the same source, encompasses a series of coordinated activities aimed at directing and controlling the fulfillment of these product requirements within a company. Thus, quality management is the systematic framework through which organizations ensure that products consistently meet established standards, enhancing overall quality and customer satisfaction.

In line with DIN EN ISO 9001 [29], quality management involves four fundamental tasks or functions designed to ensure the delivery of products that either meet or surpass customer expectations. These tasks encompass quality planning, quality assurance, quality control, and quality improvement.

- Quality planning: It establishes quality standards and specifications that products must adhere to. It involves identifying crucial quality criteria and parameters for overall measurement and evaluation. Comprehensive plans are defined, outlining the necessary processes, resources, and responsibilities to meet these quality standards. Additionally, quality planning includes a thorough risk assessment to identify potential quality risks and develop strategies for mitigation and management.
- Quality assurance: It is focused on periodic assessment and auditing of processes and systems, ensuring compliance with predefined quality standards. This task includes creating and maintaining detailed documentation, formalizing quality standards, and training employees to adhere to them. These measures collectively ensure the implementation of a systematic and structured approach to quality within the organization.
- Quality control: This task concentrates on inspecting and testing products at various production stages or at the end of the manufacturing process to ensure alignment with specified quality criteria. Quality control is also the primary focus of this work. Integral components of this task include statistical process control and root-cause analysis, facilitating ongoing processes monitoring and control while addressing the underlying causes of defects and deviations from quality standards.
- Quality improvement: It is aimed at enhancing the organization's capacity to consistently meet quality requirements. This task involves regular feedback collection from customers, employees, and stakeholders to identify areas for improvement. Continuous evaluation and enhancement of processes are

essential to increase efficiency, reduce defects, and elevate overall quality. Quality improvement also encompasses the search for innovations to refine products and processes, along with benchmarking to compare performance against industry benchmarks and best practices, thereby identifying opportunities for further improvement.

Within the broader context of quality management, as outlined by DIN EN ISO 9001[29], the pursuit of excellence involves a spectrum of coordinated activities—from planning and assurance to control and continuous improvement. Simultaneously, Schmitt et al. [94] define three critical perspectives within a manufacturing company—product, process, and system quality. These perspectives offer a comprehensive framework for evaluating and enhancing the overall quality of the manufacturing process. Figure 2.1, adapted from the work of Schmitt et al. [94], provides a visualization of these perspectives in the context of a multi-stage manufacturing process. The term 'defect slippage' in this framework refers to defective products or parts that may go unnoticed before being forwarded to subsequent processes or reaching the customer. Understanding these quality perspectives is essential for implementing effective quality control measures and ensuring the consistent delivery of products that meet or exceed customer expectations.



QG = Quality gate, FP = Final product, QI = Quality inspection

Figure 2.1.: Quality perspectives in manufacturing systems adapted from [94]

The perspective of **product quality** involves translating customers' expectations and requirements into measurable product features throughout the manufacturing process. This study concentrates on product quality, with a particular emphasis on relative measures that inform the degree of conformity with specified requirements. Parameters such as product feature tolerances, quality and defect rates, and first-pass yield are indicators for assessing product quality.

Product feature tolerances refer to acceptable variations or deviations from the nominal or target values of specific features within a product. These tolerances are precisely defined to ensure that manufactured

products align with required specifications and functional standards, accounting for inherent variability in the manufacturing process. Common types of tolerances for product features encompass:

- Dimensional tolerances: Allowable variation in the physical dimensions of a product.
- Geometric tolerances: Permissible variations in the form, orientation, and location of features. This category includes parameters such as straightness, flatness, circularity, and concentricity.
- Surface finish tolerances: Defining the texture or smoothness of a product's surface. Tolerances for surface finish outline acceptable variations in roughness or texture.
- Positional tolerances: Controlling the location of features relative to a reference point or datum. It specifies the allowable deviation from the nominal position.
- Angular tolerances: Specifying acceptable variations in angles between features. For example, if two surfaces are intended to be perpendicular, an angular tolerance outlines the allowable deviation from a perfect right angle.
- Material property tolerances: Specified for material properties such as hardness, tensile strength, or other mechanical attributes.

Tolerances play a critical role in ensuring product consistency and meeting functional requirements. Precision components may require tighter tolerances, while looser tolerances may be acceptable for less critical parts. The specific tolerances for a product depend on factors such as its intended use, manufacturing capabilities, and cost considerations. In the context of a multi-stage manufacturing process, where creating one dimension of the final part involves one or more process steps, each step introduces variations. These in-process variations are traditionally constrained by tolerances, forming a tolerance chain as outlined in the process plan [5]. In machining, the tolerance chain dictates how individual cuts aggregate to achieve each blueprint dimension [121]. The dimension resulting from a specific cut is a stochastic variable influenced by factors such as machine tool capability, raw materials, and tools. The process plan should account for these variables and distribute tolerances among the various process steps to prevent exceeding blueprint tolerance specifications as these variations accumulate and combine towards the conclusion of the manufacturing process, a phenomenon known as tolerance stack-up [121].

Quality rate is the portion of quality conform (non-defective, identified as 'OK') products divided by the total amount of produced products (i.e., Equation 2.1). A higher quality rate indicates higher product quality.

$$\text{Quality rate} = \frac{\text{OK}}{\text{OK} + \text{NOK} + \text{Defect slippage}} \tag{2.1}$$

Defect rate is the portion of non-quality conform (defective, identified as 'NOK') products divided by the total amount of produced products, as indicated by Equation 2.2. A lower defect rate indicates higher product quality.

$$\text{Defect rate} = \frac{\text{NOK}}{\text{OK} + \text{NOK} + \text{Defect slippage}} \tag{2.2}$$

First-pass yield informs the percentage of products that pass quality control in a quality gate on the first attempt without requiring rework or additional processing. It is calculated similarly to the quality rate but on a single process or quality gate level, as shown in Equation 2.3.

$$\text{First-pass yield} = \frac{\text{OK}_Q Gi}{\text{OK}_{QGi} + \text{NOK}_{QGi} + \text{Defect slippage}_{QGi}} \tag{2.3}$$

The **process quality** perspective encompasses a comprehensive examination of the production processes, machinery, and quality inspection stations involved in creating a specific product. This perspective is a measure of the extent to which a manufacturing process consistently produces products that adhere to predefined specifications and requirements. It serves as an indicator of the efficiency, reliability, and stability of the production process. Key indicators to assess process quality include the defect slippage rate at the single-process and overall production line levels. This rate reflects the extent to which defects go unnoticed within a specific process or across the entire production line. Additionally, metrics such as defect removal efficiency and process capability indices contribute to evaluating process quality. Collectively, these metrics provide insights into how effectively the production processes identify and eliminate defects, as well as the overall capabilities of the manufacturing system in maintaining consistent product quality. To improve single process quality, one can resort to the 'small control loops', as identified in Figure 2.1. Small control loops adjust process variables that directly affect the manufacturing process, e.g., machine settings. Next, the indicators for assessing process quality are presented.

Defect slippage rate is the percentage of defective products not identified as defects and wrongly forwarded to the subsequent processes (see Equation 2.4). Defect slippages increase handling, repair, and transport costs in the process chain. Besides the material waste incurred due to scrap parts, the time used in machines and workstations is also lost.

$$\text{Defect slippage rate} = \frac{\text{Defect slippage QG}_i}{\text{NOK QG}_i + \text{Defect slippage QG}_i} \tag{2.4}$$

Defect removal efficiency is the percentage of non-quality conform products that are removed before the product is delivered to the subsequent process step (i.e., Equation 2.5).

$$\text{Defect removal efficiency} = \frac{\text{NOK QG}_i}{\text{NOK QG}_i + \text{Defect slippage QG}_i} \tag{2.5}$$

Process capability indices, the Cp (Equation 2.6) and the Cpk (Equation 2.7), are statistical measures that assess the capability of a process to produce products within specified tolerance limits. A higher Cp and Cpk indicate a more capable and controlled process.

$$Cp = \frac{\text{Upper specification limit} - \text{Lower specification limit}}{6 \times \text{Standard deviation}} \tag{2.6}$$

$$Cpk = \min\left(\frac{\text{Upper specification limit} - \text{Process mean}}{3 \times \text{Standard deviation}}, \frac{\text{Process mean} - \text{Lower specification limit}}{3 \times \text{Standard deviation}}\right) \tag{2.7}$$

The **system quality** perspective encapsulates a holistic evaluation of the entire manufacturing system, considering both product and process. It encompasses the coordination, integration, and performance of all interconnected elements contributing to producing goods. Going beyond individual processes or products, system quality strives to optimize the manufacturing system's efficiency, effectiveness, and adaptability. Metrics employed to evaluate quality from this viewpoint include Overall Equipment Effectiveness (OEE), total throughput time, inventory turnover, percentage of on-time deliveries, first-pass yield, and defect slippage at the system level. One can resort to the 'large control loop' identified in Figure 2.1 to improve overall system

quality. The large control loops are used for production design, defining features, and affecting product and process quality to achieve high productivity with low inspection and rework [39].

OEE evaluates the efficiency of equipment and processes by considering availability, performance, and quality. The OEE is the product of these three factors, as shown in Equation 2.8.

$$\text{OEE} = \text{Availability} \times \text{Performance} \times \text{Quality} \tag{2.8}$$

The resulting OEE value is expressed as a percentage, with higher values indicating better overall equipment effectiveness and efficiency.

Availability measures the actual production time compared to the planned or scheduled production time (see Equation 2.9). It considers downtime, including planned stops (e.g., maintenance) and unplanned stops (e.g., breakdowns).

$$\text{Availability} = \frac{\text{Operating time}}{\text{Planned production time}} \tag{2.9}$$

Where *operating time* is the actual time the equipment is producing. *Planned production time* is the total time the equipment should be available for production.

Performance assesses the speed at which the equipment operates compared to its maximum potential speed (see Equation 2.10). It accounts for factors such as machine speed losses and suboptimal performance.

$$\text{Performance} = \frac{\text{Actual production rate}}{\text{Maximum production rate}} \tag{2.10}$$

Where *actual production rate* is the number of units produced during the operating time, *maximum production rate* is the highest achievable production rate under ideal conditions.

Quality evaluates the number of good parts produced compared to the total (i.e., Equation 2.11). It takes into account defects, rework, or scrap.

$$\text{Quality} = \frac{\text{OK}}{\text{OK} + \text{NOK} + \text{Defects slippage}} \tag{2.11}$$

The total *throughput time* is the total time required for a product to be produced, including actual processing in workstations, inspection, transport between workstations, and waiting times (i.e., Equation 2.12).

$$\text{Throughput time} = \text{Time end} - \text{Time start} \tag{2.12}$$

The *percentage of on-time deliveries* expresses the on-time delivery performance as a percentage. A higher percentage indicates a better on-time delivery performance, while a lower percentage suggests that a larger proportion of deliveries are not meeting their scheduled delivery times.

$$\text{Percentage of on-time deliveries} = \frac{\text{Number of on-time deliveries}}{\text{Total number of deliveries}} \times 100 \tag{2.13}$$

The *inventory turnover* measures how quickly a company can sell and replace its inventory. It is calculated as the ratio of the cost of goods sold to the average inventory level.

$$\text{Inventory turnover} = \frac{\text{Cost of goods sold}}{\text{Average inventory}} \tag{2.14}$$

The first-pass yield and the defect slippage on system levels are calculated similarly to the measures on the process level (Equation 2.3 and Equation 2.4), but then with the total number of OK and NOK products and the total defect slippage of the system.

The historical development of quality management [94] Quality management has evolved over time, reflecting the changes in manufacturing practices, technological developments, market and economic conditions, and a higher understanding of customer needs (see Figure 2.2). In **Pre-Industrial Revolution** times, the quality of goods relied heavily on the craftsmanship of artisans, who often performed quality inspections themselves or left the assessment to customers.

The **Industrial Revolution**, spanning the late 18th to early 19th century, brought mass production and the formalization of inspection processes, involving dedicated personnel manually evaluating products for defects. **Henry Ford** introduces the serial production and standardization through the manufacturing of the Model T.

The early 20th century witnessed the emergence of the **quality control movement** led by statisticians. Statistical models were introduced to monitor and control manufacturing processes, laying the foundations for the Statistical Process Control (SPC) and quality control charts. **Frederick W. Taylor** in the beginning of the 20th century introduces the scientific management, encompassing work division and standardization, aiming to improve efficiency. In this context, Taylor proposes the figure of an 'inspector', a specialist dedicated to the inspection of products.

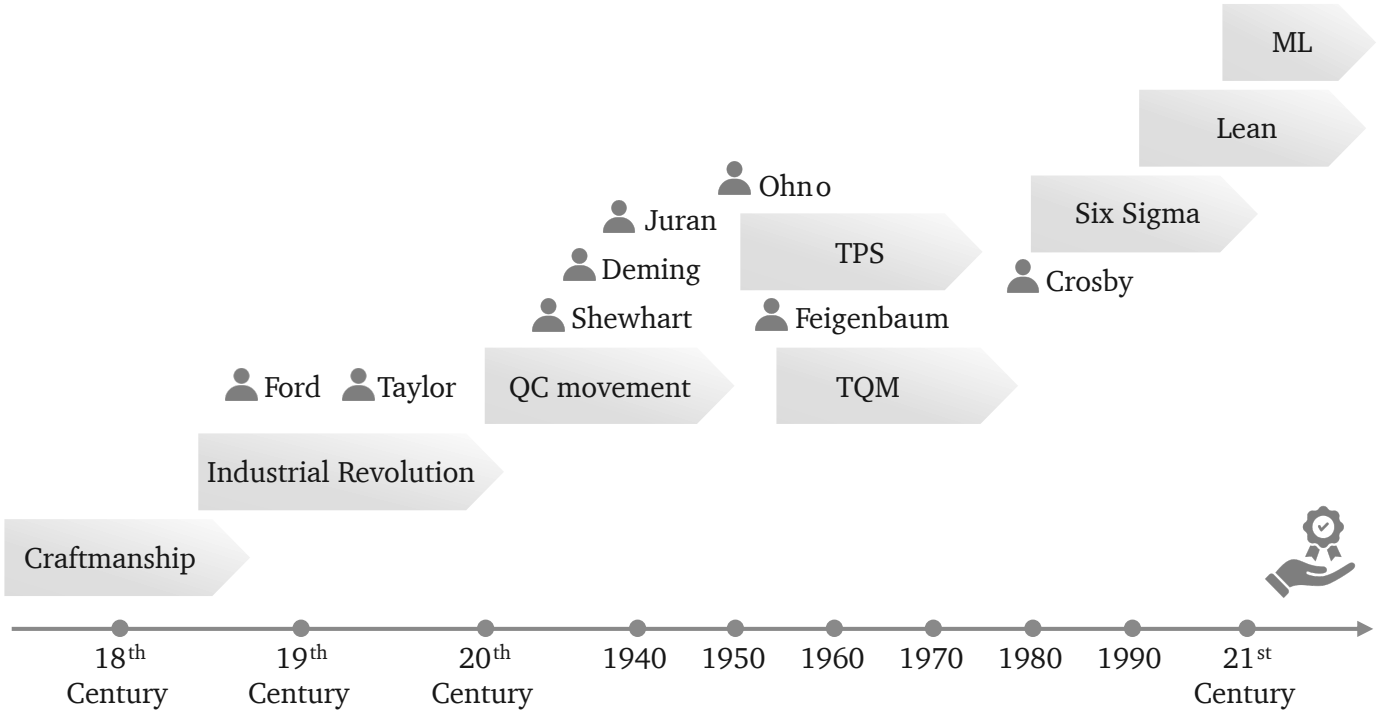


Figure 2.2.: Historical development of quality management

Walter A. Shewhart while working at Bell Telephone Laboratories publishes the book 'Economic Control of Quality of Manufactured Products' in 1931, introducing the foundations for statistical quality control. Post-World War II, quality management gained momentum, with influential figures like **W. Edwards Deming** and **Joseph M. Juran** further refining its concepts. Juran in 1951 publishes the 'Quality Control Handbook', defining the costs of poor quality and dividing them between those that are unavoidable (costs associated with prevention, inspection, sampling, sorting and further controlling processes) and those that are avoidable (costs

associated with product defects and failures, damaged material, rework and repair, customers complaints handling, and customer churn). Juran affirms that poor quality causes should be addressed in the product development process, before reaching production.

In the 1956, **Armand V. Feigenbaum** introduces the first comprehensive quality management model: the **Total Quality Management (TQM)**. The TQM approach advocates for continuous improvement efforts involving all employees, integrating quality principles and tools into a comprehensive system encompassing customer focus, continuous process enhancement, employee involvement, and statistical methods. Feigenbaum defines the quality management principles: Quality is essentially determined by the customer's expectations; every employee is responsible for quality; quality is generated by all functions in a company. According to Feigenbaum, the quality control for each product throughout its product development process can be divided in three areas:

1. Verification of the product's design regarding manufacturability and robustness of new technologies.
2. Incoming material inspection.
3. Product and production control.

The **Toyota Production System (TPS)** was developed by the automotive manufacturer Toyota in Japan in the 1940s and 1950s as a way to improve efficiency, eliminate waste, and maximize value for customers. It was a collaborative effort involving many individuals within the company. Taiichi Ohno, a Toyota executive, is often credited as one of the primary architects of the TPS. The system is based on several key principles, including:

- Continuous improvement (Kaizen): TPS emphasizes the importance of continuously seeking ways to improve processes, products, and systems. This involves empowering employees to identify and address problems, make incremental improvements, and strive for excellence.
- Just-in-Time (JIT) production: TPS aims to minimize inventory and reduce lead times by producing goods only as they are needed, in response to customer demand. This helps to eliminate waste associated with excess inventory and storage costs.
- Jidoka (Autonomation): Jidoka refers to building quality into the production process by incorporating automatic stops and error detection mechanisms. This allows workers to identify and resolve issues quickly, preventing defects from reaching customers.
- Respect for people: TPS places a strong emphasis on respect for employees, recognizing their expertise, creativity, and contributions to the organization. This involves empowering employees to participate in decision-making, providing opportunities for training and skill development, and fostering a culture of collaboration and teamwork.

The **Ishikawa diagram**, also known as the fishbone diagram or cause-and-effect diagram, was introduced by Kaoru Ishikawa, a Japanese quality control expert, in the 1960s. Ishikawa developed the diagram as a tool for identifying and visualizing the root causes of a problem or quality issue. The diagram takes the form of a fishbone, with the problem statement or effect at the head of the fish and the potential causes branching off as 'bones'. Similarly, during the same decade, the **Failure Mode and Effects Analysis (FMEA)** is introduced by the United States Armed Forces as part of the military's system reliability engineering efforts. The FMEA is a systematic approach for identifying and evaluating potential failures and defects and their sources or causes.

Philip B. Crosby wrote the book 'Quality is Free' in 1979. He later authored 'Quality Without Tears: The Art of Hassle-Free Management' in 1984 introducing the concept of 'zero defects' manufacturing. Crosby also defined the four absolutes of quality management:

1. Quality is defined as conformance to requirements: According to Crosby, quality means meeting or exceeding customer requirements and expectations. It emphasizes the importance of understanding and fulfilling customer needs to ensure satisfaction.

2. The system of quality is prevention: Crosby advocated for a proactive approach to quality management, emphasizing the importance of preventing defects rather than detecting and fixing them after they occur. This involves implementing processes and systems that minimize the likelihood of errors and defects.

3. The performance standard is zero defects: Crosby promoted the idea that organizations should strive for perfection by aiming for zero defects in their products and processes. While achieving absolute perfection may be challenging, setting this high standard encourages continuous improvement and drives organizations to minimize errors and defects.

4. The measurement of quality is the price of nonconformance: Crosby emphasized the importance of measuring the costs associated with quality issues, including the costs of defects, rework, and customer dissatisfaction. By quantifying these costs, organizations can better understand the impact of poor quality and make informed decisions to improve quality and reduce expenses.

The **International Organization for Standardization (ISO)** standardized quality management systems in 1987, introducing norms like ISO 8402 (later replaced by the ISO 9000 series in 2000), fostering systematic approaches to quality management and certification.

The 1980s also witnessed the introduction of the **Six Sigma** methodology. Six Sigma aimed to enhance business processes by defining, measuring, analyzing, improving, and monitoring them statistically, ultimately reducing defects and process variations to achieve higher performance. Six Sigma was developed by the engineer Bill Smith as a quality improvement methodology to reduce defects and improve manufacturing processes. However, it gained widespread recognition and adoption when Jack Welch implemented Six Sigma at General Electric in the 1990s, helping to popularize it as a management philosophy and business strategy. The fundamental principle of Six Sigma is to reduce variation in processes to the point where there are fewer than 3.4 defects per million opportunities. It follows a structured problem-solving approach, often referred to as DMAIC, which stands for Define, Measure, Analyze, Improve, and Control:

- Define: In this phase, the project goals and objectives are defined, along with the customer requirements. The team also identifies the process to be improved and establishes a project charter outlining the scope, timeline, and resources required.

- Measure: The second phase involves gathering data and measuring the current performance of the process. This includes identifying key metrics, collecting data, and establishing a baseline to understand the process's current state.

- Analyze: During this phase, the team analyzes the data collected to identify root causes of defects or variations in the process. Various statistical tools and techniques, such as Pareto charts, histograms, and cause-and-effect diagrams, may be used to identify areas for improvement.

- Improve: In the 'improve' phase, the focus shifts to implementing solutions to address the root causes identified in the previous phase. This may involve experimenting with process changes, implementing best practices, or redesigning workflows to achieve the desired improvements.

- Control: The final phase of DMAIC is 'control', where the team establishes controls to ensure that the improvements made are sustained over time. This includes developing monitoring systems, implementing standard operating procedures, and providing training to ensure that the process remains stable and continues to meet the desired performance levels.

Lean management, developed by a team of researchers and practitioners primarily influenced by the principles and practices of the TPS, gained prominence in the 1990s. This approach focuses on eliminating waste, emphasizing value-adding activities, and promoting continuous improvement. Lean principles focus on maximizing value for customers while minimizing waste, and they encompass concepts such as:

- Value stream mapping: Identifying and visualizing the value flow through the entire process, from raw materials to the delivery of finished products or services, and eliminating non-value-adding activities.
- Elimination of waste: Lean management aims to eliminate waste in all its forms, including overproduction, waiting time, transportation, excess inventory, motion, defects, and underutilized talent.
- Continuous flow: Creating smooth, uninterrupted flow throughout the production process to minimize delays, reduce lead times, and improve responsiveness to customer demand.
- Pull systems: Establishing pull-based systems where production is driven by customer demand, with work being initiated only when there is a demand for it.

Entering the 21st century, **Industry 4.0** brought forth transformative changes, leveraging advances in information and communication technology, automation, data analytics, and the Internet of Things (IoT). These technologies facilitated the real-time recording and monitoring of live production data, allowing a data-centric approach to quality management. This approach enhances agility and responsiveness, enabling the prediction, prevention, and real-time resolution of quality issues.

As quality management evolved, the integration of **ML** and **predictive analytics** has become increasingly prominent. ML algorithms can analyze vast datasets to identify patterns and predict potential defects. Predictive quality models enable proactive decision-making, reducing costs and time for manual quality controls. Thus, the intersection of quality management principles with ML technologies marks a significant step towards more advanced, predictive, and adaptive quality control systems in modern manufacturing.

Subsection 2.2.1 addresses predictive quality within multi-stage discrete manufacturing processes, clarifying distinctive aspects and considerations associated with this type of systems and introducing Section 2.3, which provides the necessary background on ML.

2.2.1. Predictive quality

Predictive quality, as defined by Schmitt et al. [93], relates to solutions that enable users to make data-driven predictions related to both product and process quality. Another perspective by Nalbach et al. [80] characterizes predictive quality as employing data-based methods to identify statistical patterns that forecast future developments concerning product quality. In alignment with these definitions and current advancements in the field, Tercan and Meisen [103] further define predictive quality as the utilization of ML and deep learning methods for estimating product-related quality based on process and product data to extract quality-enhancing insights. This estimation encompasses quality prediction and classifying fixed quality parameters or known product faults. In this context, process and product data encompass design factors, sensor data, production parameters, and product quality measurements. Figure 2.3 illustrates the predictive quality approach according to Tercan and Meisen [103]: Relevant process and quality data is collected from the manufacturing process and stored in a database; historical data is used for training and testing ML models; the trained model is then used to perform quality estimations for supporting decision-making carried out by human experts or other systems in production. Through this perspective, predictive quality aims to enhance quality control by replacing or reducing traditional physical product quality testing, thereby reducing costs, detecting potential failures, and providing valuable support for improving product design and manufacturing [71]. In this thesis, the focus lies on training and testing suitable ML models with historical data, as highlighted in green in Figure 2.3.

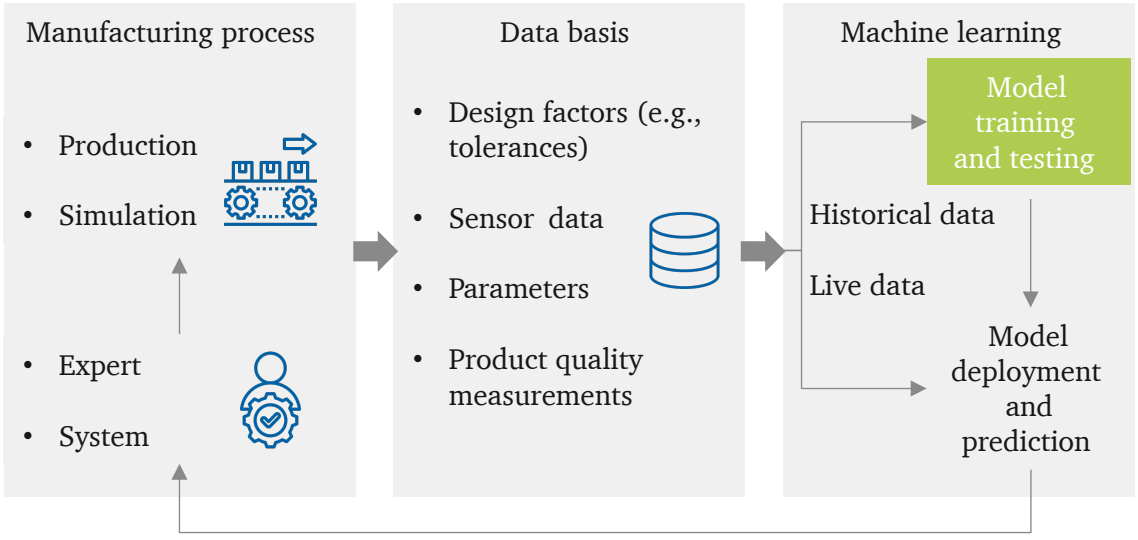


Figure 2.3.: Predictive quality approach adapted from [103]

In predictive quality literature one might often come across the term 'condition monitoring' for referring to applications with similar goals as the ones previously described. While condition monitoring primarily focuses on real-time tracking and analysis of the state of machinery and equipment [11], predictive quality extends its scope beyond machinery and equipment to encompass the quality of products produced in a manufacturing process. It involves using data-driven methods, often leveraging ML and statistical techniques, to estimate or predict product quality based on process and product data. The primary goal of condition monitoring is to detect anomalies or deviations from normal operating conditions, often in real-time, providing immediate alerts when abnormal conditions are detected. In contrast, the primary objective of predictive quality is to anticipate and enhance the quality of the final product by analyzing and understanding the relationship between process variables and product quality. While it can involve real-time analysis, predictive quality may also include predictions made at different stages of the manufacturing process to optimize and control the quality of the end product. Rostami et al. [89] identify four quality assurance and control tasks generally addressed by predictive quality applications in manufacturing:

- Quality description: Identifying and ranking factors and variables that influence product and process quality.
- Quality prediction: Developing models that predict a specific quality attribute that can be defined as real numbers.
- Parameter optimization: Finding the optimal combination of process parameters that yield a desired quality output.
- Quality classification: Classifying the product's quality into a set of pre-defined classes using process data. The classification can be with nominal, binary or ordinal quality variables.

Predictive quality holds significant potential for enhancing decision-making in quality management, spanning product, process, and system perspectives. Within the context of data-based quality control in multi-stage manufacturing systems, approaches can be categorized into two distinct types, as outlined by Arif [3]: single-point and multi-point prediction methods (refer to Figure 2.4).

- Single-point prediction methods: Single-point approaches aim to estimate the final product quality by considering data from the entire manufacturing process. These methods offer a key advantage in automating end-of-line quality controls by encompassing the entire manufacturing process when estimating or classifying product quality, considering the data of multiple workstations. A notable consideration in current research on single-point approaches is the assumption that each manufacturing workstation independently influences the final product quality. Despite their benefits, these approaches have limitations, particularly in the early detection of defects. This drawback may result in the unnecessary production of scrap products that could have been identified and addressed in preceding workstations.
- Multi-point prediction methods: Multi-point approaches shift focus to the output quality of specific workstations within the manufacturing process. An advantage of these methods lies in their potential to identify defects early in the production process, thereby preventing the propagation of faults throughout the manufacturing line. By focusing on the output quality of specific workstations, multi-point prediction methods enable the detection of quality deviations at their point of origin. However, these approaches have limitations, as individual models can only explain partial quality aspects for a particular workstation.

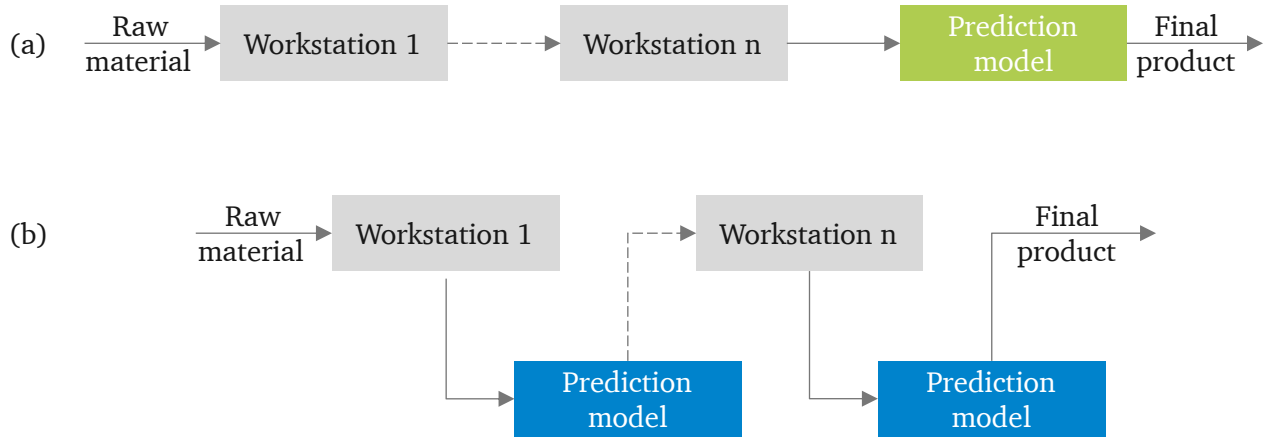


Figure 2.4.: (a) Single-point and (b) multi-point approaches according to Arif, Suryana, and Hussin [3]

The differentiation of both approaches does not exclude their potential integration within a production environment. They serve as complementary methodologies, addressing distinct quality control aspects in complex systems. This thesis aligns with single-point methods, aiming to understand product quality after multiple workstations. The primary focus, driven by the nature of the available experimental data, centers on quality classification within the context of multi-stage discrete manufacturing processes. Specifically, focusing on binary classification, distinguishing between quality-conforming products and non-quality-conforming products. Extensive research has traditionally focused on multi-point prediction methods, developing predictive quality solutions for individual machines or processes, such as turning and milling [103]. A systematic literature review is conducted in Chapter 3 to identify the methods currently used for predictive quality in multi-stage discrete manufacturing processes, thus identifying the existing gaps and drawbacks. A key objective is to tackle a drawback associated with representing interdependencies within manufacturing processes. The aim is to enhance the accuracy and robustness of quality predictions by capturing the complex relationships between different stages of production.

State-of-the-art predictive quality solutions leverage ML techniques to estimate and improve product quality. Section 2.3 introduces the basic concepts of ML and explores specific algorithms and metrics used in developing and evaluating classification models in this thesis.

2.3. Machine learning

This section draws heavily from the books titled *Understanding Machine Learning: from Theory to Algorithms* by Shalev-Shwartz et al. [97], *Pattern Recognition and Machine Learning* by Bishop [10] and *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow* by Géron [43].

ML is a subfield of the broader domain of artificial intelligence. Arthur Samuel, an American pioneer in the field of computer science and artificial intelligence, in 1959 defined ML as the 'field of study that gives computers the ability to learn without being explicitly programmed'. Tom Mitchel, professor and researcher in the field of ML, in 1997 provided a task-oriented definition:

> A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

ML is particularly suited for problems for which available solutions demand long lists of rules, complex problems in changing environments for which traditional programming approaches do not provide satisfactory solutions, and extracting insights from large amounts of data. Figure 2.5 illustrates the traditional application development approach (a) compared to the ML approach (b). The computer program's capacity to solve a particular problem depends on whether the program contains the proper set of rules. On the other hand, the ML approach uses algorithms to learn and find patterns in large and complex data. In this case, we talk about 'training' an ML algorithm.

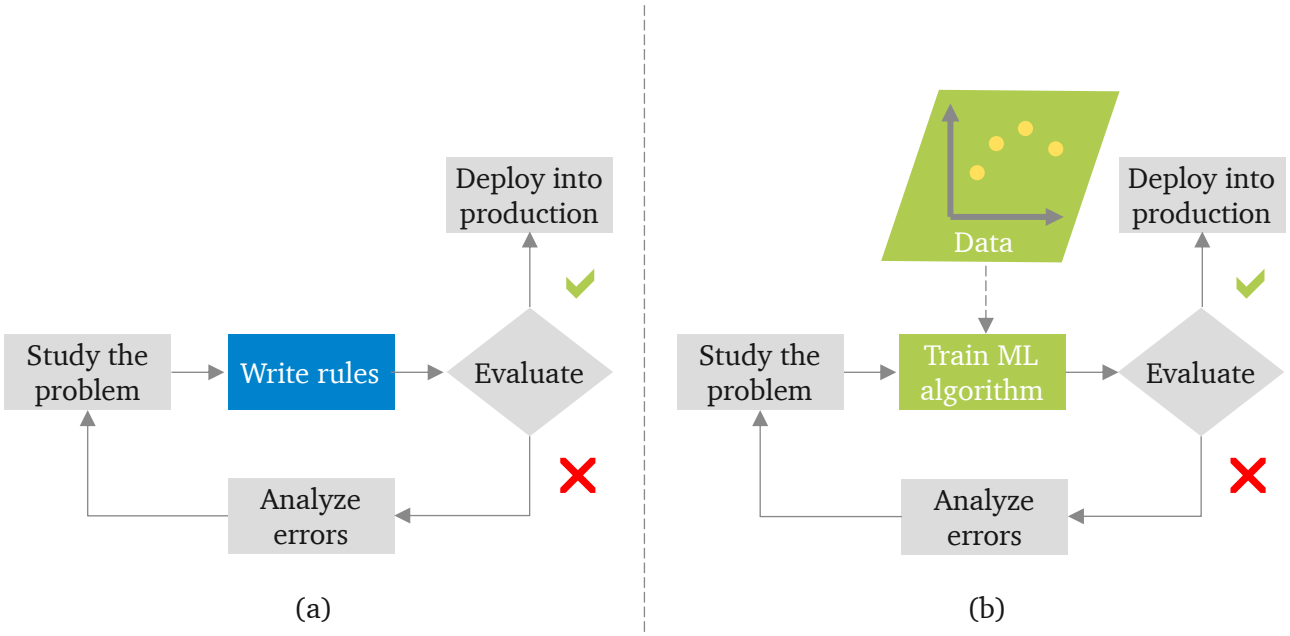


Figure 2.5.: (a) Traditional approach and (b) ML approach based on Géron [43]

ML algorithms are programming tools capable of learning from examples as they find patterns, learn these patterns and apply them to unseen data, so repetitive decisions can be automated. An example, represented as a vector $x \in \mathbb{R}^n$, is a set of features related to an object or event that the ML system should process. ML systems can be categorized based on various dimensions, such as their training mode (including supervised, unsupervised, semi-supervised, and reinforcement learning), their adaptability in learning incrementally or in real-time (online vs. batch learning), and their ability to assess new data against known data or to identify patterns and formulate predictive models (instance-based vs. model-based learning) [43].

Supervised learning techniques, the focus of this work, are used to estimate a numerical (regression) or categorical (classification) target variable based on selected input variables. In supervised learning, the training set encompasses the sought solutions called *labels* or *targets*. Supervised learning algorithms observe examples $\mathbf{x}$ and the associated label y and learn to predict y from $\mathbf{x}$, usually by estimating $p(y|\mathbf{x})$. This work centers around a specific supervised learning task known as binary classification. In this type of task, the computer program is tasked with predicting the category to which a given data item belongs, with only two possible classes. Section 2.3.2 provides a more detailed explanation of classification tasks and the algorithms used to solve them.

In supervised learning, the algorithm is provided with a labeled dataset, where each example consists of input features $\mathbf{x}$ and their corresponding correct output or labels y. Generally speaking and particularly for artificial neural networks, training this type of ML algorithm involves the following steps:

1. Divide the dataset: Divide the dataset into training, validation, and test sets to facilitate effective model development and evaluation. This division helps ensure the reported performance metrics are reliable indicators of the model's ability to generalize to new, unseen data.

2. Initialize model: Select a supervised learning model architecture suitable for the task.

3. Define the loss function: Define a loss function (or cost function) that quantifies the difference between the model's predicted output and the actual output labels in the training data. The loss function is a measure of how well the model is performing.

4. Initialize model parameters: Initialize the model parameters randomly (or with some predefined values).

5. Optimize: Use an optimization algorithm (e.g., Stochastic Gradient Descent (SGD)) to adjust the model parameters iteratively. The optimization process minimizes the loss function by finding the optimal parameters that make the model's predictions closer to the actual labels. During training, the algorithm goes through multiple iterations, or epochs, of the training process. The model processes the entire training dataset in each epoch, makes predictions, calculates the loss, and updates its parameters.

6. Tune hyperparameters and validate model: Periodically assess the model's performance on a separate validation set not seen during training. Validation helps monitor how well the model generalizes to new, unseen data and enables adjustments to hyperparameters to prevent overfitting. Hyperparameters are configuration settings that are not learned from the data but are set before the training begins. Unlike model parameters learned during training, hyperparameters are predefined and play a crucial role in controlling the learning process. During hyperparameter tuning, hyperparameters are tested and selected based on the model's performance on the validation set to optimize the training process and improve the model's generalization.

7. Test model: After training, evaluate the model's performance on a separate test set not encountered during training or validation. Testing provides a final, unbiased assessment of the model's ability to make predictions on new, unseen data and estimate how well the model is likely to perform in real-world scenarios.

Loss function Specifically for binary classification tasks, loss functions quantify the difference between the predicted class probabilities (or scores) and the actual class labels for each data item. The goal is to define a measure the algorithm can minimize during training. Commonly used loss functions for binary classification are the binary cross-entropy loss, the Hinge loss, squared Hinge loss, and logistic loss.

The Binary Cross-Entropy Loss (BCEL) (also called log loss) is suitable for binary classification problems where y_i is the actual label (0 or 1) of the i-th instance, p_i is the predicted probability that the instance belongs to class 1, and n represents the total number of instances in the dataset:

$$\text{BCEL} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)] \tag{2.15}$$

The Hinge loss is commonly used in Support Vector Machine (SVM)[115]. $f(x_i)$ is the raw decision function output, and y_i is the true class label (-1 or 1). The loss is zero when the predicted score for the correct class is greater than or equal to 1, and it increases linearly as the prediction becomes less confident:

$$\text{Hinge Loss} = \max(0, 1 - y_i \cdot f(x_i)) \tag{2.16}$$

The squared Hinge loss penalizes misclassifications more heavily than the Hinge loss. This stronger penalization can be beneficial for applications where a more decisive penalty for misclassification is wanted:

$$\text{Squared Hinge Loss} = \max(0, 1 - y_i \cdot f(x_i))^2 \tag{2.17}$$

The logistic loss is used in Logistic Regression (LR). $f(x_i)$ is the raw output of the logistic regression model, and y_i is the actual class label (-1 or 1). The loss penalizes the model more for confidently incorrect predictions:

$$\text{Logistic Loss} = \log(1 + e^{-y_i \cdot f(x_i)}) \tag{2.18}$$

Optimization algorithms Optimization algorithms adjust the input variables $\mathbf{x}$ of a function $f(\mathbf{x})$ to find a set of optimal model parameters θ that minimize or maximize this function. The function being minimized or maximized, known as the objective function, is alternatively referred to as the cost function, loss function, or error function in the case of minimization.

Gradient descent is an optimization algorithm that finds the minimum of a function by iteratively moving in the direction of the steepest descent, guided by the negative gradient of the function. The gradient is the vector containing all the partial derivatives of the function, denoted as $\bigtriangledown_{\mathbf{x}} f(\mathbf{x})$, where element i of the gradient is the partial derivative of f with respect to $\mathbf{x_i}$ and measures how the function changes as the variable $\mathbf{x_i}$ increases at point $\mathbf{x}$. The model parameters θ are updated by moving in the opposite direction of the gradient:

$$\theta = \theta - \epsilon \cdot \nabla f(\theta) \tag{2.19}$$

Where ϵ is the learning rate, a hyperparameter, and a scalar value that determines the step size in the parameter space. The negative sign in Equation 2.19 ensures the algorithm descends toward the minimum. The learning rate is crucial in determining the convergence speed and stability of the algorithm. Choosing an appropriate learning rate is essential; a too-small rate may result in slow convergence, while a too-large rate may lead to oscillations or divergence. This step is repeated until a stopping criterion is met. Standard stopping criteria include reaching a maximum number of iterations, achieving a certain level of convergence, or observing minimal changes in the objective function.

Stochastic Gradient Descent (SGD) is an extension of gradient descent that deals with the problem of computing the gradient using the entire dataset in each iteration for large datasets, which can become computationally expensive and impracticable. SGD randomly selects a subset (mini-batch) of the data for each iteration, reducing computation time while still providing a good approximation of the gradient.

Alternative optimization algorithms, such as Adam, RMSprop, and Adagrad, incorporate additional techniques to adapt the learning rates dynamically or address challenges like saddle points (points with zero slope that have neighbors that are both higher and lower than the point itself) in the loss landscape.

Overfitting and underfitting When training an ML model with examples, the collection of available examples with the respective labels is called a dataset. Training and test sets are disjoint subsets of the dataset. As the name suggests, the training set is used for the optimization task of learning model parameters through examples contained in the training set. Subsequently, the model's performance is assessed with new, previously unseen data during inference on the test set.

In this context, the concept of generalization comes into play, representing the model's ability to perform well when presented with inputs it has not encountered during training. This aspect distinguishes ML from mere optimization. The ultimate objective is to achieve a low generalization error, a metric measured on the test set that reflects the expected error when processing new input data. During the model training process, a set of critical assumptions, known as the i.i.d. (independent and identically distributed) assumptions, are employed to ensure the representativeness of the training data compared to the data the model will encounter in deployment. These assumptions assert that examples within each dataset are independent, and both the training and test sets are drawn from an identical probability distribution. The objective during training is to minimize the training error and the gap between training and test errors. This goal leads to two central challenges in ML: underfitting and overfitting. *Underfitting* happens when the model cannot obtain a low error on the training set. Underfitting indicates that a model is too simple to capture the underlying patterns in the training data, failing to learn the relationships between features and labels. Conversely, *overfitting* occurs when the discrepancy between training and test errors is too large. Overfitting indicates that a model is overly complex and learns the noise and fluctuations in the training data rather than the underlying patterns. As a result, it performs well on the training set but poorly on new, unseen data. A model with *appropriate capacity* will neither underfit nor overfit - the model is neither too simplistic to grasp the complexities of the data nor excessively complex to overfit and memorize noise.

Using polynomial regression, Figure 2.6 showcases the distinctions between underfitting, appropriate capacity, and overfitting with synthetically generated data. In the underfitting scenario, the model is too simplistic to capture the patterns within the data. Represented by a linear fit (polynomial degree = 1), the straight line fails to capture the curvature present in the dataset. This oversimplified model leads to poor performance, overlooking essential features and nuances. An appropriate level of model capacity reaches a balance, capturing the essential patterns without unnecessary complexity. In this case, a quadratic fit (polynomial degree = 2) represents the data's general trend. The model neither oversimplifies nor overcomplicates, demonstrating its capacity to generalize to new, unseen data while accurately reflecting the training dataset. The model exhibits excessive capacity in the overfitting scenario (shown in the rightmost plot), fitting the training data too closely. A polynomial fit of degree 7 follows each data point, capturing even the noise and fluctuations present. However, this hyper-specific fit fails to generalize to new data, highlighting the model's overemphasis on patterns that may not hold beyond the training set.

The concept of model capacity and finding the appropriate capacity leads to a theorem introduced by David Wolpert in 1997, the so-called *no free lunch theorem* [111]. This theorem says there is no one-size-fits-all or universally better algorithm or strategy that performs optimally across all possible problems. In other words, there is no superior algorithm that outperforms all others across diverse problem domains. The theorem thus implies that the performance of an algorithm is contingent on the specific characteristics of the problem at hand. The theorem does not imply that all algorithms are equal; instead, it suggests that the superiority of an algorithm is problem-specific:

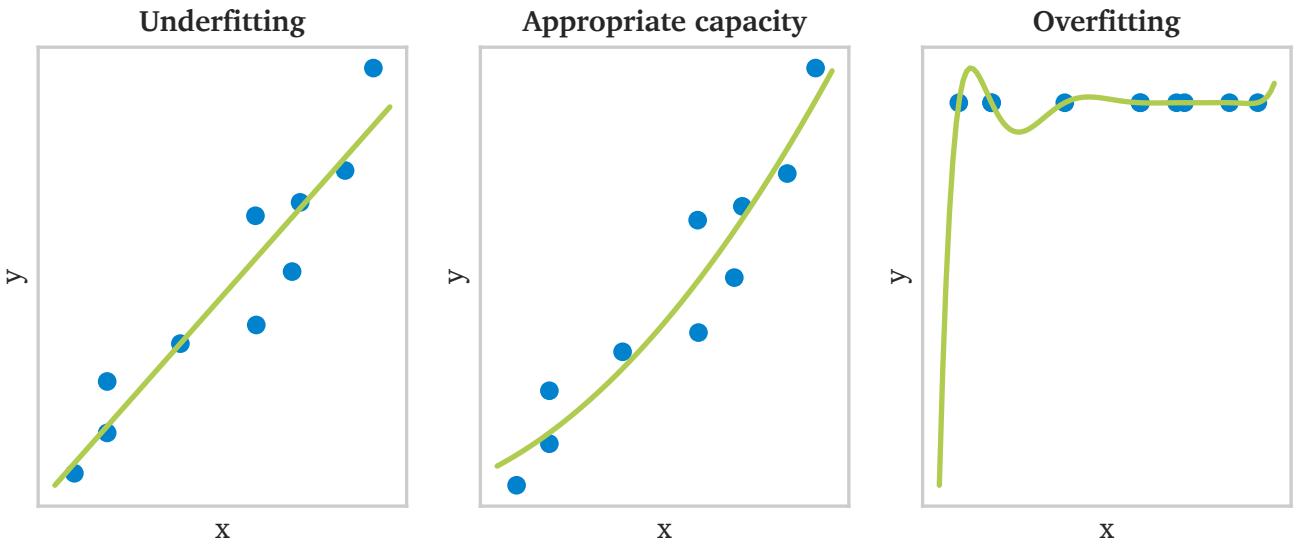


Figure 2.6.: Underfitting, appropriate capacity and overfitting examples

- No algorithm is universally superior for all possible problem instances. An algorithm that performs well on one type of problem may perform poorly on another.
- The performance of an algorithm is tied to the nature of the problem it is designed to solve. Different algorithms may excel in different problem domains, and trade-offs often involve choosing one algorithm over another.
- The effectiveness of an algorithm depends on the assumptions and characteristics of the problem space. What works well in one context may not generalize to other contexts.
- It is essential to consider and experiment with various algorithms to find an algorithm that performs well on a specific problem. No algorithm is guaranteed to be the best across all scenarios.

A model's representational capacity describes thus the complexity and flexibility in learning and representing patterns within the data, and it is defined by its hypothesis space. A hypothesis space is the set of all possible hypotheses, or candidate functions the algorithm can use to model the relationship between input and output. Changing the model's hypothesis space and applying **regularization techniques** to indicate preferences for specific functions can reduce the model's generalization error and achieve better solutions for a given problem. Section 2.3.2 presents the baseline ML algorithms applied in this work. Two types of regularization techniques are often employed: L1 regularization (Lasso) and L2 regularization (Ridge).

In *L1 regularization*, a penalty term is added to the objective function proportional to the model parameters' absolute values. It encourages sparsity in the model, meaning that some model parameters may become exactly zero, leading to feature selection. L1 regularization is particularly useful when there is a suspicion that many features are irrelevant or redundant.

In *L2 regularization*, a penalty term is added to the objective function proportional to the squared values of the model parameters. It penalizes large parameter values, preventing any feature from largely influencing the model. L2 regularization effectively prevents multicollinearity (high correlation between features) and stabilizes the model.

The regularization term is typically added to the standard loss function that the model minimizes during training. The overall objective function combines the original loss function and the regularization term. A hyperparameter controls the strength of regularization, often denoted as λ, which determines the trade-off between fitting the training data and keeping the model parameters small. The regularized objective function (J) is often expressed as:

$$J(\theta) = \text{Original loss} + \lambda \times \text{Regularization term} \tag{2.20}$$

Where θ represents the model parameters. Regularization is a tool for improving model generalization by preventing models from becoming too complex and overfitting the training data. The choice of the regularization type and strength depends on the specific characteristics of the data and the model.

In the upcoming Section 2.3.1, the focus shifts to feature engineering—an integral component of data analysis and ML.

2.3.1. Feature engineering

This section's content is derived from the *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists* by authors Zheng and Casari [124] and *Feature Engineering for Machine Learning and Data Analytics* by authors Dong and Liu [36]. Throughout this section, unless otherwise indicated by a specific citation, the text draws from the insights and knowledge presented in the books previously mentioned.

The performance of ML algorithms and their capacity to extract meaningful patterns heavily depends on the representation of the input data [44]. In this context, *feature* is an attribute or variable used to describe some aspect of the data. Features can be of various types, such as numerical (quantitative and continuous), ordinal and categorical values, text, images, and audio signals. Informative and expressive features capture the relevant information in the data and generate accurate predictive models that learn the meaningful patterns in the data. The degree to which a feature is informative or expressive is measured in terms of the improvement it brings to the task at hand.

Feature engineering encompasses the extraction and selection of relevant features from the raw data into suitable formats to improve the performance of ML models. Next, the most relevant feature extraction and selection techniques for this work are introduced with a special focus on time series data, since these features will be extracted in one of the presented case studies.

Feature extraction Feature extraction (also known as feature transformation and generation) refers to the construction of new features from the existing raw data. Feature transformation is usually achieved using mathematical mappings, such as calculating the arithmetic mean. Feature generation requires not only the original data but also domain knowledge, as in the case of creating binary or numerical representations of categorical variables.

Statistical features. Statistical features coming from the descriptive statistics domain can be divided in central tendency (or centrality) and variation (or variability) measures. These measures capture certain properties of a dataset by summarizing and aggregating it [99]. Centrality measures extract information about the data distribution's center:

- *Arithmetic mean* or also called *average*, denoted by $\bar{x}$, is the sum of values divided by the number of observations:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{2.21}$$

- *Geometric mean* is the *n*th root of the product of *n* values:

$$\prod_{i=1}^{n}{}^{1/n} = \sqrt[n]{x_1 x_2 ... x_n} \tag{2.22}$$

- *Median* is the middle value of a dataset when it is ordered is ascending order. The median is particularly useful when dealing with datasets that have skewed distributions or outliers, as it is not influenced by the specific numerical values in the extremes of the dataset, unlike the *mean*. It provides a measure of central tendency that is less affected by outliers.

- *Mode* is the value that appears most frequently in the dataset. A dataset can have no mode (if no value is repeated more often than other values), one mode or multiple modes (if more than one value has the highest frequency).

Variability measures extract information about the data spread, *i.e.* how far the measurements are from the center:

- *Standard deviation*, denoted by σ, is the sum of squares differences between each value in the dataset and the mean:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}} \tag{2.23}$$

- *Variance*, denoted by σ^2, is the square of the *standard deviation*:

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1} \tag{2.24}$$

- *Range* is the difference between the minimum and maximum values in a dataset.

- *Interquartile range* is the range of values that span the middle 50% of the dataset. It is the difference between the third quartile (75th percentile) and the first quartile (25th percentile).

- *Skewness* quantifies the asymmetry of a probability distribution. For a random variable X, the skewness is defined as:

$$Skewness = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{\sigma}\right)^3 \tag{2.25}$$

  Where $\bar{x}$ is the mean , and σ is the standard deviation of the distribution. The sign of the skewness indicates the direction of the skew. A positive skewness indicates a right-skewed distribution. A negative skewness indicates a left-skewed distribution. A skewness equal to zero indicates a perfectly symmetric distribution.

- *Kurtosis* describes the 'tailedness' or shape of the probability distribution. It provides information about the concentration of values in the distribution's tails. For a random variable **x**, the kurtosis is defined as:

$$Kurtosis = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{\sigma}\right)^4 - 3 \tag{2.26}$$

  Where $\bar{x}$ is the mean, and σ is the standard deviation of the distribution. A high kurtosis indicates heavy tails, meaning extreme values are more likely. In contrast, low kurtosis indicates extreme examples are less likely, and most values are concentrated closer to the center of the distribution.

Temporal features. Temporal features are measures derived from the timing or sequence of events in time series data. These can include the time of the day (expressed in hour, minute or second information), day of the week, month or season and customized lag features that take the previous observations as features. These features are derived from the temporal ordering of observations over time and are essential for understanding and modeling time-dependent phenomena. Examples of temporal features and their use are described below:

- *Timestamps* are date and time information of when the observations are recorded. They form the basis for establishing the temporal order of the data.
- *Time intervals* are the duration between consecutive timestamps, which can reveal patterns related to seasonality, frequency of observations, or changes in data collection frequency.
- *Temporal lags* are defined based on the time series values at previous time points. For example, lag-1 represents the value at the previous time step, lag-2 represents the value two-time steps ago, and so on. Lag features help capture temporal dependencies and autocorrelation.
- *Moving averages* are the average of a subset of recent observations over a specified window. Moving averages help in smoothing out short-term fluctuations and highlighting long-term trends.
- *Time of day/week/month/year* features contain information about the time of day, day of the week, month, or year associated with each observation.
- *Temporal aggregations* are built by aggregating data over different intervals to derive summary statistics. For example, daily, weekly, or monthly averages.
- *Temporal deltas* are the differences between consecutive observations that highlight the time series' rate of change or acceleration.

Frequency domain features. Frequency domain features provide information about the frequency components present in the data. These features are derived through techniques such as Fourier or Wavelet transform. Extracting frequency domain features is particularly useful for understanding the periodicity, seasonality, and dominant frequencies in time series data.

The *Fourier Transform* is a mathematical operation that decomposes a function or signal in the time domain into its constituent frequencies in the frequency domain. The continuous Fourier Transform of a signal $x(t)$ is defined as follows:

$$X(f) = \int_{-\infty}^{\infty} x(t) \cdot e^{-j2\pi ft}\, dt \tag{2.27}$$

Where $X(f)$ is the frequency representation of the signal in the frequency domain, $x(t)$ is the original signal in the time domain, f is the frequency variable, and j is the imaginary unit.

The inverse operation, known as the inverse Fourier Transform, can reconstruct the original signal from its frequency representation:

$$x(t) = \int_{-\infty}^{\infty} X(f) \cdot e^{j2\pi ft}\, df \tag{2.28}$$

In practice, the Continuous Fourier Transform is often replaced by its discrete counterpart, the Discrete Fourier Transform (DFT), which is computationally more efficient and suitable for digital signal processing. The DFT of a discrete signal $x[n]$ of length N is given by:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi}{N}kn} \tag{2.29}$$

Where $X[k]$ is the frequency representation of the signal at frequency index k, $x[n]$ is the discrete signal at time index n, and N is the length of the signal. The Fast Fourier Transform (FFT) algorithm is used to compute the DFT efficiently, making it practical for real-time and large-scale applications.

The *Wavelet Transform* is used in signal processing for analyzing and representing signals in terms of localized patterns called wavelets. Unlike the Fourier Transform, which represents a signal using sinusoidal basis functions that extend infinitely, the Wavelet Transform uses wavelets localized in time and frequency. This localization allows the Wavelet Transform to capture and analyze transient features and changes in a signal with high precision.

The Continuous Wavelet Transform (CWT) of a signal $x(t)$ is defined as follows:

$$X(a,b) = \int_{-\infty}^{\infty} x(t) \cdot \psi^* \left(\frac{t-b}{a} \right) dt \tag{2.30}$$

Where $X(a,b)$ is the wavelet transform of the signal at scale a and translation b, $x(t)$ is the original signal in the time domain, $\psi(t)$ is the analyzing wavelet, which is typically a function with zero mean and finite energy, a is the scale parameter that controls the width of the wavelet, and b is the translation parameter that shifts the wavelet along the time axis. The asterisk ψ^* denotes the complex conjugate.

In practical applications, the CWT is often discretized to form the Discrete Wavelet Transform (DWT), which is computationally more efficient and allows working with discrete signals. The DWT involves a process of successive approximation and downsampling, resulting in a multiresolution analysis of the signal. The DWT is derived from the CWT by discretizing both the scale and translation parameters. Considering a discrete signal $x[n]$ sampled at equally spaced points in time. The DWT is obtained by applying a series of filtering and downsampling operations. The signal $x[n]$ is successively passed through two filters: a low-pass filter $h[n]$ and a high-pass filter $g[n]$. This results in two sets of coefficients: approximation coefficients (representing the low-frequency components) and detail coefficients (representing the high-frequency components). The DWT of a discrete signal $x[n]$ at scale o and position k is given by:

$$\begin{aligned} A_{o,k} &= \sum_{m=-\infty}^{\infty} x[m] \cdot h[m-2k] \\ D_{o,k} &= \sum_{m=-\infty}^{\infty} x[m] \cdot g[m-2k] \end{aligned} \tag{2.31}$$

Where $A_{o,k}$ represents the approximation coefficients and $D_{o,k}$ represents the detail coefficients at scale o and position k. The parameter o indicates the level of decomposition, while k denotes the translation parameter. After obtaining the approximation and detail coefficients, the signal is downsampled by a factor of 2, discarding alternate samples. This process is repeated iteratively, leading to a hierarchical decomposition of the signal into different frequency bands. The DWT thus provides a sparse representation of the original signal, capturing both its frequency content and localization in time.

Examples of frequency domain features calculated either with the Fourier or Wavelet Transforms are presented below:

- *Periodogram/peak frequencies:* The periodogram plots the spectral density of a time series against frequency. Peak frequencies in the periodogram correspond to dominant frequencies in the data. The frequency corresponding to the highest peak can be considered a significant frequency component.

- *Power Spectral Density (PSD):* It represents power distribution across different frequencies in the time series. Peaks in the PSD indicate dominant frequencies and the total power measures signal strength.
- *Dominant frequency:* It is the frequency at which the maximum power or amplitude occurs. It is often identified by locating the peak frequency in the periodogram or PSD.
- *Harmonic frequencies:* They are frequencies that are integer multiples of the dominant frequency. These can reveal patterns of periodicity and harmonics in the data.
- *Spectral entropy:* It is a measure of the distribution of spectral power across different frequencies. It quantifies the complexity or randomness of the frequency components in the time series.
- *Wavelet Transform coefficients:* Features can be derived from the Wavelet Transform coefficients at different scales and positions.
- *Autocorrelation at lag frequencies:* Autocorrelation measures the similarity between a signal and a delayed version of itself. The autocorrelation function can be computed at different lag frequencies to capture periodic patterns.
- *Cross-correlation:* It measures the similarity between two time series at different time lags. Cross-correlation in the frequency domain helps identify relationships between different frequency components in two time series.

Feature selection Feature selection, as the name suggests, is the selection of a subset of the most relevant and expressive features. It is important to remove irrelevant or redundant features that slow down training or introduce noise or bias in the predictor thus reducing classification performance. This subsection is based on the papers "A survey on feature selection methods" from Chandrashekar et al. [19] and "A comprehensive survey on feature selection in the various fields of machine learning" from Dhal et al. [33].

Feature selection techniques can be classified into filter, wrapper, and embedded models. *Filter models* rank features according to the dependent variable, so only the features above a certain threshold are selected and applied to the prediction task. A relevant feature (feature above the defined threshold) should contain useful information to discriminate different classes in the data. Filter models, therefore, evaluate the feature subset according to some quality measurement technique independent of the predictor (the classification algorithm).

Wrapper models evaluate the performance of the predictor with each set of features, i.e., the predictor is wrapped in a search algorithm which finds the features that lead to the highest performance. This means wrapper models use the classification algorithm as performance measure. The wrapper approach adds and removes features from the subset and observes the resulting change in performance. Feature selection is done in two phases: First, using the training dataset, the optimal or best performing feature subset is generated based on the classifier's performance. Second, the classifier is trained with the selected feature subset and evaluated using the test dataset.

Embedded models perform feature selection as part of the training process, aiming to reduce the computation time taken to test different subset of features as performed in wrapper methods. This is achieved for example through regularization techniques, such as L1 regularization (Lasso), which penalizes the coefficients of irrelevant features, driving them to zero and eliminating them from the model. Decision tree-based algorithms also have built-in mechanisms to evaluate feature importance during training. The filter approach is faster than wrapper and embedded models, since the classifier is only evaluated after the feature selection is done and not iteratively for different feature subsets. Wrapper methods are computationally expensive because they involve training the model multiple times for different feature subsets. Embedded methods are computationally more efficient than wrapper methods as feature selection is integrated into the model training process. Table 2.1

provides an overview of the different models, including their assessment criteria, search strategy, strengths and weaknesses.

Table 2.1.: Comparison between filter, wrapper, and embedded model approaches adapted from [33]

Approach	**Filter**	**Wrapper**	**Embedded**
Assessment criteria	Statistical test	Cross-validation	Cross-validation
	Feature subset relevance	Feature subset usefulness	Feature subset usefulness
Search	Through the order of the features (via nested feature subsets or by the individual ranking of features)	Search all possible feature subsets	The learning process controls the search
Strengths and weaknesses	Not sensitive to the algorithm choice	Sensitive to the algorithm choice	Sensitive to the algorithm choice
	Univariate feature selection The selection of useful features can fail	It selects the most useful features but has massive time complexity	Comparatively lower time complexity

In the experiments conducted in this thesis, various ML algorithms are evaluated and compared. Filter methods provide a more fair basis for comparison. By conducting feature selection prior to cross-validation, a uniform dataset and feature set can be established for training and testing all algorithms. This approach ensures consistency and fairness in the comparison process across different models. Next, the filter-based feature selection technique used in this thesis is described.

Statistical feature selection assesses the relevance of features to the target variable. A univariate feature significance test is performed for each feature in the input feature matrix X and the target vector y. These tests yield p-values, indicating the statistical significance of each feature's impact on the target variable [25]. Two hypotheses are tested [106]:

H_0 : The feature lacks relevance and should not be considered.

H_1 : The feature is relevant and should be retained.

For binary target variables:

H_0 : The distribution of the feature is equal across each target class.

H_1 : The distribution of the feature differs across target classes.

In essence, this feature selection approach evaluates the importance of features by conducting statistical tests, calculating p-values, and ordering the features by relevance for selecting the most significant features relevant to the target variable. In the case of a binary target variable and continuous features, the standard statistical test to assess the difference in the distribution of each feature between the two classes is the Mann-Whitney U test, also known as the Wilcoxon rank-sum test.

The Mann-Whitney U test is a non-parametric statistical test used to assess whether two independent samples come from populations with the same distribution or if one population tends to have larger values

than the other. It is commonly used when the assumptions of parametric tests like the t-test are not met, such as when the data are not normally distributed or when the sample sizes are small. This test is conducted through the following steps:

1. Ranking: Combine the data from both samples and rank them in ascending order. Assign ranks to the data, with the smallest value receiving a rank of 1, the second smallest a rank of 2, and so on. If there are ties (i.e., identical values), assign the average rank to each tied value.
2. Sum of ranks: Calculate the sum of ranks for each sample. Let U_1 be the sum of ranks for sample 1 and U_2 be the sum of ranks for sample 2.
3. U-statistic: Calculate the Mann-Whitney U statistic, which is the smallest of U_1 and U_2. The formulas for calculating the U statistic are:

$$U_1 = n_1 \cdot n_2 + \frac{n_1 \cdot (n_1 + 1)}{2} - \sum_{i=1}^{n_1} R_{1i} \tag{2.32}$$

$$U_2 = n_1 \cdot n_2 + \frac{n_2 \cdot (n_2 + 1)}{2} - \sum_{i=1}^{n_2} R_{2i} \tag{2.33}$$

   Where n_1 is the number of observations in group 1, n_2 is the number of observations in group 2, R_{1i} is the rank of the i-th observation in group 1, R_{2i} is the rank of the i-th observation in group 2, U_1 is the U statistic for group 1 and U_2 is the U statistic for group 2.

$$U = \min(U_1, U_2) \tag{2.34}$$

4. P-value: Once the U statistic is calculated, the p-value associated with it can be calculated. The p-value represents the probability of observing a U statistic as extreme as the one calculated if the null hypothesis were true, i.e., if there were no difference between the distributions of the two samples.
   - For small sample sizes (typically n_1 and n_2 less than 20), typically the exact method would be used to calculate the p-value.
   - For larger sample sizes, an approximation based on the normal distribution can be used. The p-value can be calculated using the normal approximation:

$$p = 1 - \Phi\left(\frac{U - \hat{x}_U}{\sigma_U}\right) \tag{2.35}$$

     Where Φ is the cumulative distribution function of the standard normal distribution, $\hat{x}_U$ and σ_U are the mean and standard deviation of the U distribution, respectively.

The null hypothesis H_0 of the Mann-Whitney U test states that there is no difference between the distributions of the two samples, while the alternative hypothesis H_1 suggests that one population tends to have larger values than the other. A small p-value indicates that the difference in the distributions is statistically significant, leading to rejection of the null hypothesis.

Section 2.3.2 introduces classification algorithms for predictive quality. With a set of features acquired through extraction and selection techniques, the exploration extends to the methodologies employed for categorizing and assigning labels to data points. The following section thus explains the principles behind selected classification algorithms, investigating how they leverage features to make predictions.

2.3.2. Classification

Quality control in multi-stage discrete manufacturing, particularly for end-of-line testing, revolves around distinguishing between defective (non-quality conform) and non-defective (quality conform) products. This task can be effectively approached as a binary classification problem. In the context of this classification task, an algorithm determines the class membership y' of an unidentified data item $\mathbf{x}'$ by analyzing a dataset $D = (\mathbf{x_1}, y_1), ..., (\mathbf{x_n}, y_n)$ consisting of data items $\mathbf{x_i}$ (input variables) with their corresponding known class memberships y_i [37]. In binary classification, the class labels y assume either 1 or 0 values (typically labeled as the positive class, denoted as 1, and the negative class, denoted as 0), providing a clear delineation between defective and non-defective products.

Classification algorithms are of two primary types based on their approach to assigning labels to new data items. The first type employs a dual distinction, assigning 0 or 1 to unknown data items. Support Vector Machine (SVM) exemplify this approach, delineating a clear boundary between the two classes. In contrast, the second type models the probability distribution $P(y|\mathbf{x})$. This approach not only results in a class label for a data item but also provides a probability of class membership. Algorithms following this approach, such as Logistic Regression (LR), Random Forest (RF), k-Nearest Neighbor (kNN), and Multilayer Perceptron (MLP), offer a more nuanced understanding by quantifying the likelihood of an instance belonging to a particular class. This probabilistic modeling enables these algorithms to capture and express uncertainty in the classification process. Both types of classifiers will be explained in further detail in this section.

Quality control is focused on minimizing false positives (non-defective items incorrectly identified as defective) and false negatives (defective items missed). Classification models provide the flexibility to fine-tune the threshold for prediction probabilities, aligning error metrics with the specific quality control objectives. This ability to optimize error metrics enhances the overall effectiveness of the quality control process.

Logistic Regression (LR) LR is an algorithm that uses the logistic function (also known as the sigmoid function) to model the probability that a data item belongs to the positive class (label equals to 1). The logistic function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.36}$$

Where $z = b + w_1x_1 + w_2x_2 + \ldots + w_nx_n$ is the linear combination of the input features x_i, b is the bias term and w_i the weights. This function transforms the linear combination z into a value between 0 and 1, representing the predicted probability of the input belonging to the positive class. The decision boundary is defined by a threshold (commonly set to 0.5) that converts the probability into binary predictions. If the predicted probability is above the threshold, the data item is classified as belonging to the positive class; otherwise, it is classified as belonging to the negative class.

An LR model is trained by providing a training dataset of data items $\mathbf{x}_i$ and the respective labels y_i, the weights and bias are adjusted during training to minimize the difference between the predicted probabilities and the actual class labels [37]. In LR, regularization is used to avoid overfitting and achieved by adding a penalty term to the cost function that is minimized during training [75]. The two common types of regularization in LR are L1 regularization (Lasso, shown in Equation (2.37)) and L2 regularization (Ridge, shown in Equation (2.38)).

$$J(w) = BCEL + \frac{\lambda}{2n} \sum_{j=1}^{m} |w_j| \tag{2.37}$$

$$J(w) = BCEL + \frac{\lambda}{2n} \sum_{j=1}^{m} w_j^2 \tag{2.38}$$

Where w represents the model parameters, BCEL is the Binary Cross-Entropy Loss defined in Equation 2.15, n is the number of examples, m is the number of features, and λ is the regularization parameter. In both regularization types, the regularization parameter λ controls the strength of regularization. A higher λ value leads to stronger regularization, which can help prevent overfitting by discouraging the model from assigning too much importance to any particular feature. Algorithm 1 describes the training and prediction process for an LR model.

Algorithm 1: LR training and prediction

Data: Training dataset: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$
Result: Trained LR model

1 **Initialization:** Set learning rate ϵ, regularization parameter λ, and initialize weights **w**;

2 **Optimization: while** *Not converged* **do**
3 Compute the logistic loss and its gradient with respect to weights;
4 Update weights using gradient descent with regularization;

5 **Trained Model:** **w** are the learned weights of the logistic regression model;

6 **Prediction:**
Data: Test instance $\mathbf{x}_{\text{test}}$
Result: Compute the logistic function to get the probability $p(y = 1|\mathbf{x}_{\text{test}}, \mathbf{w})$;
Make predictions based on the threshold (e.g., 0.5) or use class probabilities;

Random Forest (RF) An RF is an algorithm introduced by Breiman [14] as a combination of decision tree predictors (also called an ensemble). The author defines it as 'a classifier consisting of a collection of tree-structured classifiers $t(\mathbf{x}, \Theta_k), k = 1, ..., n$ where the Θ_k are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input **x**.' The term 'independent' in this context means the parameters Θ_k associated with each tree are not influenced by each other. In other words, the construction of one tree in the forest does not depend on the construction of any other tree. Each tree is built independently of the others. 'Identically distributed' means that the probability distribution governing the generation of the parameters Θ_k is the same for all trees in the forest. In other words, each tree is drawn from the same distribution of parameters. The parameters Θ_k are referred to as random vectors because they consist of multiple random variables (elements of the vector) that collectively determine the structure and behavior of each decision tree in the forest. This ensures consistency and fairness in the construction of each tree. The basic idea behind an RF is the combination of multiple weak learners (decision trees) to produce a robust model that generalizes well to unseen data.

The RF uses bootstrapped sampling, the process of creating multiple subsets of the original dataset through random sampling with replacement, for selecting a random subset of the training data to train each tree (individual decision tree classifier). At each node of the decision tree, a random subset of features is split. Multiple decision trees are then trained on a different subset of data and features. In classification tasks, the class that is output most frequently by the individual trees is considered as the final prediction: $\hat{y} = \text{mode}(\{t(\mathbf{x}) \mid t \in T\})$, where T is the set of all trees in the forest and $\hat{y}$ the final prediction.

An individual decision tree classifier has a root node, decision nodes, and leaf nodes [21]. The algorithm starts with the whole dataset as the root node. It selects a dataset feature to split it based on specific criteria, such as Gini impurity. Each subset of the dataset corresponds to a different branch of the node. The feature selection and dataset splitting are repeated recursively, creating new nodes and branches until a stopping condition, such as tree depth and minimum number of samples in a node, is met. The decision nodes are the

nodes between the root and leaf nodes. Leaf nodes contain the predicted output for the corresponding subset of data. The classification for a new data item is done by traversing the decision tree from the root node and following the branches according to the feature values of the data item until it reaches a leaf node. The objective is to build decision trees with feature splits that result in subsets with homogeneous classes, which are evaluated by the impurity measure chosen to assess the quality of the splits. Gini impurity quantifies the likelihood of incorrectly classifying a randomly chosen item of the dataset if it were randomly labeled according to the distribution of labels in the subset. The Gini impurity for a binary classification task is given by:

$$\text{Gini}(p) = 1 - p_0^2 - p_1^2 \tag{2.39}$$

Where p_0 and p_1 represent the proportions of instances belonging to class 0 and class 1 in the node, respectively. Algorithm 2 describes the training and prediction process for an RF model.

Algorithm 2: RF classifier training and prediction

Data: Training dataset: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$
Result: Trained RF model

Initialization: Set the number of trees T, and other hyperparameters;

Training: **for** $t = 1$ **to** T **do**
    Sample a bootstrap dataset from the training data;
    Train a decision tree on the bootstrap dataset;
    Store the trained tree in the forest;

Prediction:
Data: Test instance $\mathbf{x}_{\text{test}}$
for $t = 1$ **to** T **do**
    Use the t-th tree to predict the class for $\mathbf{x}_{\text{test}}$;

Ensemble: Aggregate individual tree predictions (e.g., by majority voting) to get the final prediction;

Support Vector Machine (SVM) The SVM algorithm is based on four key concepts: the separating hyperplane, the maximum-margin hyperplane, the soft margin, and the kernel function [81]. For SVM, data items for classification are treated as points in a high-dimensional space, and the separating hyperplane is a line in this space that distinguishes between the data items. The maximum-margin hyperplane is chosen to maximize the distance from the given feature vectors representing the samples' characteristics or attributes in a classification task. The margin is defined as the distance from the separating hyperplane to the nearest expression vector.

The soft margin is a hyperparameter that balances hyperplane violations and margin size. It allows for flexibility by permitting some data points (anomalous expressions) to fall on the other side of the separating hyperplane without significantly affecting the final result.

The kernel function projects the data into a higher-dimensional space, making it linearly separable. The kernel function is a hyperparameter that can be optimized accordingly, aiming to separate the data without introducing unnecessary dimensions [81].

SVM assumes that the data used for training and testing are drawn from the same distribution without assuming a specific class of distributions. The SVM algorithm utilized in this work is from the Scikit-learn library, specifically the C-Support Vector Classification. For a set of training vectors $\mathbf{x_i} \in \mathbb{R}^n, i = 1, ..., n$,

belonging to two classes and an indicator vector $\mathbf{y} \in \mathbb{R}^l$ such that $y_i \in 1, -1$, the algorithm solves the following optimization problem [20]:

$$\begin{aligned} \min_{\mathbf{w},b,\varepsilon} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\phi(\mathbf{x_i}) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \quad i = 1, \ldots, l, \end{aligned} \tag{2.40}$$

Where $\phi(\mathbf{x_i})$ maps $\mathbf{x_i}$ into a higher-dimensional space and $C > 0$ is the regularization parameter which controls the trade-off between achieving a low training error and a simpler decision boundary. The slack variable ξ_i allows for some degree of misclassification. The constraints ensure the examples are correctly classified or lie on the correct side of the decision boundary. The kernel function $\phi(\mathbf{x_i})$ enables the model to handle non-linear decision boundaries. In summary, the goal of the optimization problem is to find the optimal hyperplane separating the training data into two classes while allowing for some degree of misclassification. Algorithm 3 describes the training and prediction process for an SVM model.

Algorithm 3: SVM classifier training and prediction

Data: Training dataset: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$
Result: Trained SVM model parameters: $\mathbf{w}$, b

1 **Initialization:** Set learning rate ϵ, regularization parameter C, and initialize $\mathbf{w}$, b, and $\boldsymbol{\xi}$;

2 **Optimization: while** *Not converged* **do**
3 Update $\mathbf{w}$ and b using gradient descent on the objective function;
4 Update $\boldsymbol{\xi}$ based on the SVM constraints;

5 **Prediction:**
Data: Test instance $\mathbf{x}_{\text{test}}$
Result: Make predictions with the trained model, where $\mathbf{w}$, b are learned parameters of the hyperplane that separates the classes;

k-Nearest Neighbor (kNN) The algorithm kNN is based on the nearest neighbor principle, which finds a predefined number of training samples closest in distance to the new data item, and predicts the label from these. 'k' in this case, indicates the number of neighbors that are being considered. kNN is a type of instance-based learning, non-parametric or non-generalizing learning: It stores instances of the training data to compare with the new data points without trying to construct a general internal model. The classification class is defined by a majority vote: The point is assigned to the class which has the most representatives within the nearest neighbors of the point.

The nearest neighbors of a data item are defined in terms of the standard Euclidean distance [6]. The distance between two data items $\mathbf{x_i}$ and $\mathbf{x_j}$ is denoted as $d(\mathbf{x_i}, \mathbf{x_j})$ and defined as:

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n}(a_r(x_i) - a_r(x_j))^2} \tag{2.41}$$

Where $a_r(x)$ is the value of the rth attribute of the data item $\mathbf{x}$. Given a data item $\mathbf{x_q}$ to be classified, let $\mathbf{x_1}, ..., \mathbf{x_k}$ denote the k instances from the training examples that are the nearest to $\mathbf{x_q}$, the kNN returns:

$$\hat{y_q} = \arg\max_{v \in V} \sum_{k=1}^{X} \delta(v, y_i) \tag{2.42}$$

Where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise. The value $\hat{y}(x_q)$ is the mode of the true labels y among k training examples nearest to $\mathbf{x_q}$. Algorithm 4 describes the training and prediction process for a kNN model.

Algorithm 4: kNN training and prediction

Data: Training dataset: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$
Result: Predicted class for a test instance $\mathbf{x}_{\text{test}}$

Training: Store the training dataset;

Prediction:
Data: Test instance $\mathbf{x}_{\text{test}}$, number of neighbors k
for $i = 1$ **to** n **do**
    Compute the distance between $\mathbf{x}_{\text{test}}$ and $\mathbf{x}_i$;
Sort the distances in ascending order and select the top k neighbors;
Result: for $j = 1$ **to** k **do**
    Aggregate the class labels of the k neighbors (e.g., by majority voting);

eXtreme Gradient Boosting (XGBoost) XGBoost, similar to RF, constructs an ensemble of weak learners (individual decision trees) and optimizes them to minimize a specific loss function [40]. The basic idea behind the algorithm is to iteratively add weak learners to correct the errors of the previous models. The objective function is a combination of a loss function (for binary classification, the logistic loss) and a regularization term:

$$\text{Objective} = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{2.43}$$

Where:

n is the number of training samples,
y_i is the true label of the i-th sample (either 0 or 1),
$\hat{y}_i$ is the predicted probability of the positive class for the i-th sample,
$\ell(y_i, \hat{y}_i)$ is the logistic loss,
K is the number of weak learners (trees),
f_k is the k-th weak learner,
$\Omega(f_k)$ is the regularization term.

The predicted probability $\hat{y}_i$ for a data item is obtained by summing the predictions from the individual decision trees and applying a sigmoid function to convert the sum into a probability, which is then used to make the binary classification:

$$\hat{p}(y = 1|x) = \frac{1}{1 + e^{-\sum_{k=1}^{K} f_k(x)}} \tag{2.44}$$

The regularization term penalizes complex models to avoid overfitting, and it is defined as:

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{2.45}$$

Where:

T is the number of leaves in the tree,
w_j is the score associated with the j-th leaf,
γ and λ are regularization parameters.

Algorithm 5 describes the training and prediction process for an XGBoost model.

Algorithm 5: XGBoost training and prediction

Data: Training dataset: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$
Result: Trained XGBoost model

Initialization: Set the number of boosting rounds T, learning rate ϵ, and other hyperparameters;
Initialize the ensemble model with a constant value (e.g., mean of the target variable);
Boosting Rounds: for $t = 1$ **to** T **do**
    Compute the negative gradient of the loss function with respect to the current model's predictions;
    Fit a weak learner (e.g., decision tree) to the negative gradient;
    Update the ensemble model by adding the weak learner with a scaled weight;
Trained Model: The final ensemble model represents the trained XGBoost model;
Prediction:
Data: Test instance $\mathbf{x}_{\text{test}}$
Result: Make predictions by summing the predictions of all weak learners in the ensemble;

Multilayer Perceptron (MLP) An MLP, also called feedforward neural network, aims to approximate a function $y = f^*(\mathbf{x})$ that maps an input $\mathbf{x}$ to a category y. It defines a mapping $y = f(\mathbf{x}, \theta)$ and learns the parameter values θ that provide the best function approximation [44]. MLPs are called feedforward networks because of how information flows and the lack of feedback connections.

MLPs are called networks because they can be represented as a directed acyclic graph composed of different functions, for example, three functions, $f(1)$, $f(2)$, and $f(3)$, connected in a chain, forming $f(x) = f_1(f_2(f_3(x)))$. In this example, function f defines a layer of the network. The final layer, the output layer, should return a value close to the desired label, y. The hidden layers are those between the first and the output layer.

The functioning of an MLP can be understood from the more basic concept of a perceptron model. The perceptron model illustrated in Figure 2.7 receives as input $\mathbf{x} = \{x_1, x_2, \ldots, x_m\}$, which represents the incoming signals and outputs a single binary value. Each input neuron x_i connects to the perceptron via a link whose strength is represented by a weight w_i. Inputs with higher weights have a more significant influence on the perceptron's output. The perceptron first computes the weighted sum z, also called net input, of the incoming signals by multiplying each input by its corresponding weight, where w_0 is the bias term:

$$z = \sum_{i=0}^{m} w_i x_i \tag{2.46}$$

The perceptron then applies an activation function to the weighted sum to generate the output. The perceptron model can be viewed as a single-layer neural network with a step function as the activation function. MLPs, on the other hand, can have multiple layers and non-linear activation functions, which allow them to learn complex, non-linear relationships in data.

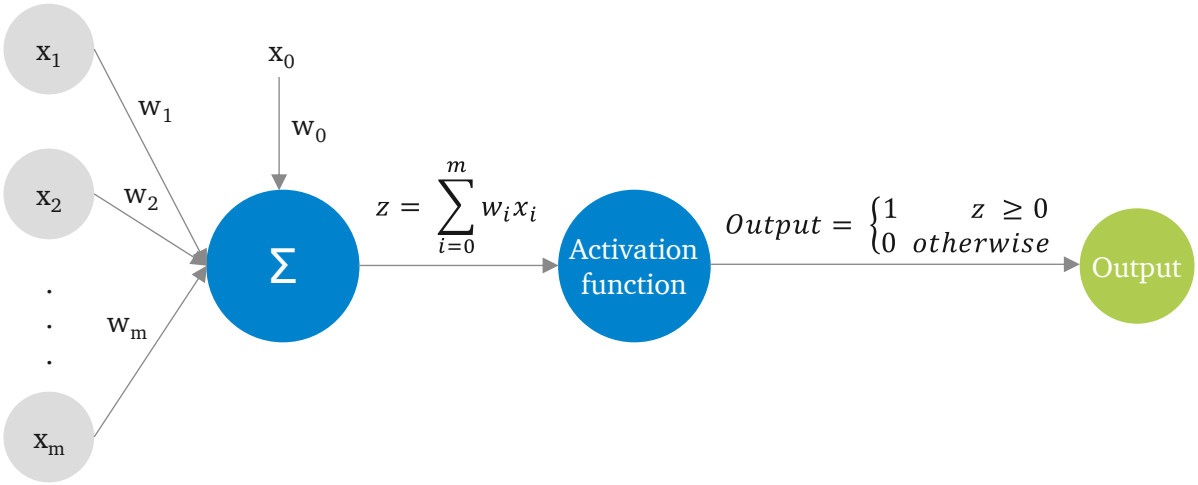


Figure 2.7.: Perceptron model

Activation functions are mathematical operations applied to each neuron (or node) output in a layer. Commonly used activation functions are the sigmoid, hyperbolic tangent, rectified linear unit, leaky rectified linear unit, and exponential linear unit.

Sigmoid The sigmoid activation function (Equation 2.47) compresses the input values to the range between 0 and 1, making it suitable for binary classification problems. However, it suffers from the vanishing gradient problem, when the gradients of the loss function with respect to the parameters (weights) become small as they are backpropagated through the network during the training process, which can lead to slow or stalled learning.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.47}$$

Hyperbolic Tangent (Tanh) The Tanh activation function (Equation 2.48) is similar to the sigmoid but maps input values to the range between -1 and 1. It helps somewhat mitigate the vanishing gradient problem but may still suffer from it.

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.48}$$

Rectified Linear Unit (ReLU) The ReLU activation function (Equation 2.49) replaces negative values with zero, introducing non-linearity and allowing the network to learn complex patterns. The values range from 0 to infinite. However, it can suffer from the 'dying ReLU' problem, where neurons can become inactive and stop learning during training.

$$\text{ReLU}(x) = \max(0, x) \tag{2.49}$$

Leaky Rectified Linear Unit (Leaky ReLU) The leaky ReLU activation function (Equation 2.50) addresses the dying ReLU problem by allowing a small, non-zero gradient for negative values, preventing neurons from becoming inactive.

$$\text{Leaky ReLU}(x) = \max(\alpha x, x), \quad \text{where } \alpha \text{ is a small positive constant.} \tag{2.50}$$

Exponential Linear Unit (ELU) The ELU activation function (Equation 2.51) is another alternative to ReLU that handles negative values more smoothly, reducing the risk of dead neurons. The values range from minus infinite to infinite.

$$\text{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}, \quad \text{where } \alpha \text{ is a positive constant.} \tag{2.51}$$

The decision to use one activation function or another depends on the problem's specific characteristics, the neural network's architecture (not only restricted to MLPs), and considerations such as the potential for vanishing or exploding gradients during training.

MLPs are trained with a learning algorithm called **back-propagation**, or backward propagation of errors. Back-propagation starts with the forward pass in which an input is fed into the network, and each layer applies a weighted sum and activation function to compute the output. The network's final output is compared to the actual label, and the loss is calculated. The backward pass propagates the error backward through the network to compute the gradients of the loss with respect to the weights. The forward, loss calculation and backward passes are repeated for multiple iterations, allowing gradual adjustment of the weights to minimize the loss function during training. Algorithm 6 describes the training and prediction process for an MLP model.

Algorithm 6: MLP classifier training and prediction

Data: Training dataset: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$
Result: Trained MLP model

Initialization: Set the number of hidden layers, number of neurons per layer, learning rate ϵ, and other hyperparameters;
Initialize the weights and biases of the neural network randomly;
Forward Pass: for $i = 1$ **to** n **do**
    Perform the forward pass to compute the predicted output for each instance;
Backward Pass (Backpropagation): while *Not converged* **do**
    Compute the loss between predicted and true labels;
    Perform the backward pass to compute gradients with respect to weights and biases;
    Update weights and biases using gradient descent or other optimization algorithms;
Trained Model: The final trained MLP model is represented by the learned weights and biases;
Prediction:
Data: Test instance $\mathbf{x}_{\text{test}}$
Result: Perform the forward pass to obtain the predicted output for the test instance;

The effectiveness of the presented classification algorithms can only be fully assessed by evaluating their performance using appropriate metrics. Classification metrics provide valuable insights into the accuracy, precision, recall, F_1 score, and other performance indicators of classification models, enabling researchers to

assess their predictive power and make informed decisions about model selection and refinement. Section 2.3.3 presents the classification metrics used in this thesis.

2.3.3. Classification metrics

Classification metrics for binary classification tasks evaluate how well a classifier correctly distinguishes between two classes. The classification metrics presented in this section are directly or indirectly used to evaluate the performance of the proposed approach compared to the identified baseline algorithms. This section's content is derived from "An overview of general performance metrics of binary classifier systems" by author Raschka [87] and "Binary classification performance measures/metrics: A comprehensive visualized roadmap to gain new insights" by authors Canbek et al. [18].

Confusion matrix The confusion matrix reports the four base performance measures for a binary classifier, namely the true positive (TP), false positive (FP), false negative (FN), and true negative (TN) predictions. Each matrix row represents the predicted class, and each column represents the actual class, as Figure 2.8 depicts.

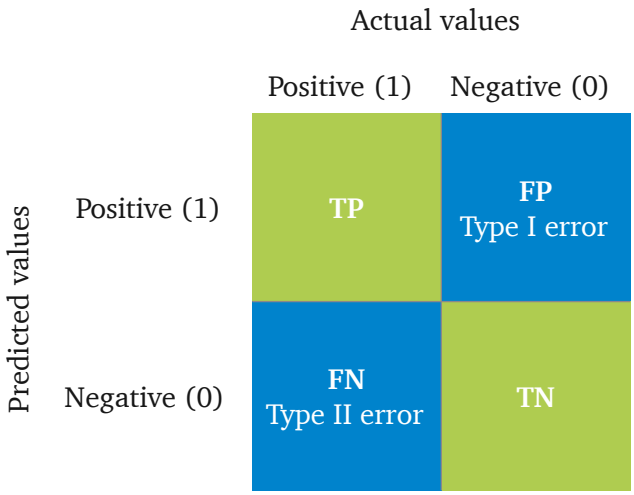


Figure 2.8.: Confusion matrix

Where:

- TP is the number of instances of the positive class correctly classified as positive by the model.
- TN is the number of instances of the negative class correctly classified as negative by the model.
- FP, also known as Type I error, is the number of instances of the negative class incorrectly classified as positive by the model.
- FN, also known as Type II error, is the number of instances of the positive class incorrectly classified as negative by the model.

Accuracy (ACC) and Prediction Error (ERR) The ACC is the sum of correct predictions divided by the total number of predictions:

$$\text{ACC} = \frac{TP + TN}{FP + FN + TP + TN} = 1 - \text{ERR} \tag{2.52}$$

The ERR is the sum of all false predictions divided by the number of total predictions:

$$\text{ERR} = \frac{FP + FN}{FP + FN + TP + TN} = 1 - \text{ACC} \tag{2.53}$$

The ACC and, consequently, the ERR values range from 0 to 1. A perfect model (which makes no incorrect predictions) has an ERR of 0 and an ACC 1. A model with an ERR of 1 and ACC of 0 makes no correct predictions.

These metrics can be misleading and provide an overly optimistic view when evaluating the performance of ML models with highly imbalanced datasets. In an imbalanced dataset, if one class is dominant, a model that predicts the majority class for all instances can achieve high ACC (and low ERR) even if it fails to predict any instance of the minority class correctly. They do not distinguish between different types of errors, and they might give a false impression of a well-performing model.

True Positive Rate (TPR) and False Positive Rate (FPR) The TPR, known as sensitivity or recall, and FPR, known as the probability of false alarm, measure the proportion of actual positive instances the model correctly identifies. TPR values range from 0 to 1. A high TPR indicates that the model is effective at identifying positive instances and is particularly relevant in cases where the cost of missing positive instances is high:

$$\text{TPR} = \frac{TP}{P} = \frac{TP}{FN + TP} \tag{2.54}$$

FPR measures the proportion of actual negative instances incorrectly identified as positive by the model. FPR values range from 0 to 1. A low FPR indicates that the model has a low rate of incorrectly classifying negative instances as positive and is essential in cases where false alarms or false positives are costly.

$$\text{FPR} = \frac{FP}{N} = \frac{FP}{FP + TN} \tag{2.55}$$

Receiver Operator Characteristic (ROC) and Area Under the Receiver Operator Characteristic Curve (AUC-ROC) The ROC curve shows the trade-off between TPR and FPR at various threshold settings for a binary classifier. The curve is obtained by plotting the TPR against the FPR for different classification thresholds.

The AUC-ROC is a scalar value between 0 and 1, where 0.5 corresponds to random guessing and 1 to perfect classification, which quantifies the overall performance of the binary classifier. The AUC-ROC , as the name suggests, is calculated as the area underneath the ROC curve. A higher AUC-ROC value indicates a better overall classifier performance across different thresholds.

Figure 2.9 shows ROC curves for two classification models, labeled model 1 and model 2. The x-axis corresponds to the FPR, and the y-axis corresponds to the TPR, also known as sensitivity or recall. The ROC curves illustrate the trade-off between true positive rate and false positive rate across different decision thresholds. A blue curve represents model 1, and a green curve represents model 2. The AUC-ROC is indicated in the legend for each model, serving as a quantitative measure of the models' discriminative ability. Additionally, a gray dashed line represents the ROC curve for random guessing. The plot visually

demonstrates the relative performance of the two models in terms of their ability to distinguish between positive and negative classes. A model with a higher AUC-ROC generally indicates better overall performance in classification. In this case, the AUC for model 2 appears to be higher than that of model 1, suggesting that model 2 outperforms model 1 regarding discriminative accuracy. AUC-ROC is used when the class distribution is balanced or when the cost of false positives and false negatives is roughly equal.

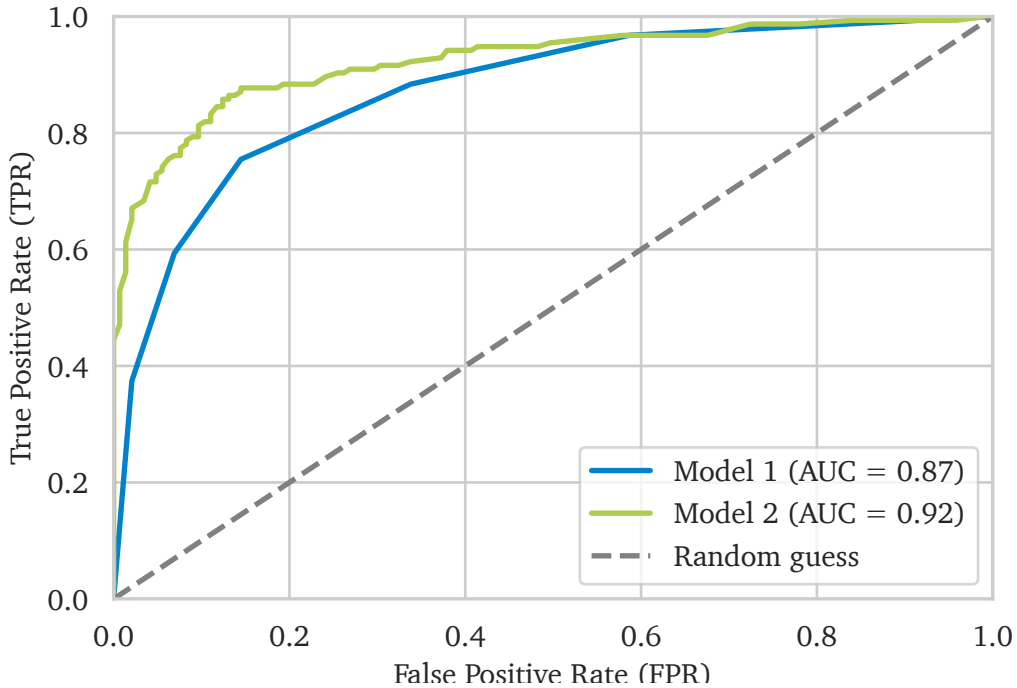


Figure 2.9.: Examples of ROC curves

Precision, recall, $\mathbf{F_1}$ score and precision-recall curve Precision measures the accuracy of positive predictions made by the model. It is particularly relevant when the cost of false positives is high. A high precision indicates that it is likely to be correct when the model classifies an instance as part of the positive class.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.56}$$

Precision values range from 0 to 1. A precision of 1 means that every instance predicted as positive is positive (no false positives exist). A precision of 0 means every instance predicted as positive is incorrect (no true positives exist).

Recall, also called sensitivity or TPR, has already been presented. Recall measures the ability of the model to identify all positive instances. It is essential when the cost of false negatives is high.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.57}$$

Recall values range from 0 to 1. A recall of 1 means that every positive instance is correctly predicted (no false negatives exist). A recall of 0 means the classifier fails to identify any positive instance (no true positives exist).

Figure 2.10 displays precision-recall curves for two different classification models, denoted model 1 and model 2. Precision is represented on the y-axis, and recall (or sensitivity) is on the x-axis. Precision-recall curves illustrate the trade-off between precision and recall across various decision thresholds. A blue curve depicts model 1, while a green curve represents model 2. The area under the precision-recall curve (AUC) is shown in the legend for each model, serving as a quantitative measure of the models' ability to balance precision and recall. Higher AUC values generally indicate better performance in terms of precision-recall trade-offs. The plot visually demonstrates the two models' relative effectiveness in identifying positive instances while minimizing false positives. Model 2 has a higher AUC, suggesting it achieves a better balance between precision and recall than model 1. Precision-recall curves are particularly useful when dealing with imbalanced datasets, where the class of interest is less frequent, and the focus is on correctly identifying positive instances while minimizing false positives.

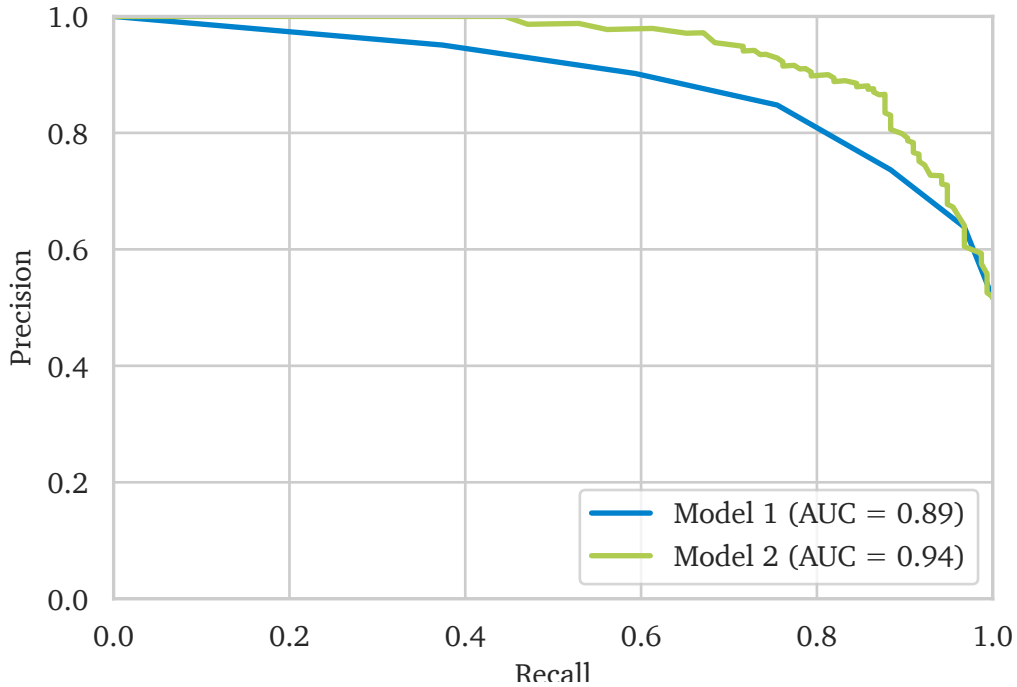


Figure 2.10.: Examples of precision-recall curves

The F_1 score is the harmonic mean of precision and recall, thus providing a balanced measure suitable for problems with imbalanced class distribution. A higher F_1 score indicates a better trade-off between precision and recall.

$$\mathrm{F}_1 \text{ score} = \frac{2 \cdot \mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} \tag{2.58}$$

The F_1 score can assume values between 0 and 1. An F_1 score of 0 indicates that either precision or recall is zero, meaning the classifier failed to identify positive instances. An F_1 score of 1 indicates perfect precision and recall. The classifier has successfully identified all positive instances without any false positives.

Matthews Correlation Coefficient (MCC) The MCC metric was formulated by Brian W. Matthews, a British biophysicist and structural biologist, in 1975 [74]. He introduced this metric as a measure of the quality of

binary classifications in the context of predicting protein secondary structure. Matthews recognized the need for a metric that would provide a balanced evaluation of classification performance, especially when dealing with imbalanced datasets. This metric considers the TP, TN, FP, and FN to calculate a balanced measure of classification performance:

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.59}$$

The MCC can assume values from -1 to +1, where +1 indicates perfect prediction, 0 indicates no better than random prediction, and -1 indicates total disagreement between prediction and observation.

2.4. Graph representation learning

As seen in Section 2.3.1, the effectiveness of ML techniques heavily relies on the algorithms' design and a good representation (feature set) of the data. Graphs are data structures that can be used to describe complex systems, such as social networks, interactions between drugs and proteins, atoms in a molecule, and connections between terminals in a telecommunication network [47]. A central hypothesis of this thesis is that a multi-stage manufacturing process can be effectively represented as a graph. Subsequently, this data representation can be utilized as input for an ML algorithm for performing a quality classification task. The innovation within this thesis involves adapting and applying a technology already employed in disciplines such as social network analysis and biology to the manufacturing domain, demonstrating the potential advantages that manufacturing companies can derive from this emerging trend.

Representation learning goes beyond mapping the representation to the output, it uses ML to discover the data representation itself [44]. Graph representation learning, also known as the process of uncovering underlying data features that facilitate the extraction of valuable information for downstream tasks, such as classification [125], involves leveraging inductive bias to emphasize specific patterns in the data, thereby enhancing model performance [7]. Deep learning-based representation learning is a process within the field of ML where the goal is to automatically learn meaningful and hierarchical representations of data through the use of deep neural networks [113]. Each layer of the neural network extracts increasingly complex features, allowing the model to learn hierarchical and abstract representations of the data. Key characteristics of deep learning-based representation learning include:

- Automatic feature learning: The model autonomously discovers relevant features from the raw input data without explicit feature engineering.
- Hierarchical representations: Deep architectures enable the learning of hierarchical and abstract representations by stacking multiple layers of neural units.
- End-to-end learning: The entire model, including the representation learning part, can be trained end-to-end using backpropagation and optimization techniques.
- Versatility: Deep learning-based representation learning can be applied to various types of data, such as images, text, sequences, and graph-structured data.

When applied to graph-structured data, representation learning incorporates relational inductive biases into deep learning architectures, allowing the development of systems capable of learning, reasoning, and generalizing from such complex data structures [47]. Section 2.4.1 provides the foundations of graph theory and necessary concepts for understanding graph representation learning and ML on graph data. Section 2.4.2 introduces the concept of GNN and how it can be used for performing end-to-end classification tasks.

2.4.1. Graph theory and machine learning on graph data

A graph, G, is composed of a node set V and an edge set E, where nodes represent entities and edges represent the relationship between entities $G = (V, E)$. Graphs can be mathematically represented through an *adjacency matrix* $\mathbf{A} \in \mathbb{R}^{|V|\times|V|}$. In the adjacency matrix, the nodes in the graph are ordered so that every node indexes a row and column in the matrix. The existence of an edge between two nodes are denoted as entries in this matrix: $A[u, v] = 1$ if $(u, v) \in E$ and $A[u, v] = 0$, otherwise. If the graph only contains undirected edges, then A is a symmetric matrix, but if the graph is directed then A is not necessarily symmetric. For graphs with weighted edges, the entries of the adjacency matrix are arbitrary real-values.

In addition to revealing the connections between nodes through edges, the graph structure can convey additional information by associating attributes with nodes, edges, or the entire graph [113]. Graph attributes, also called *features*, are represented as vectors associated with each node or edge in the graph. A heterogeneous graph (or heterograph) besides the node set V and edge set E, is also composed of a node type mapping function $\eta\colon V \to S$, and an edge type mapping function $\varphi\colon E \to R$, where S and R denote the node and edge types, where $|S| + |R| > 2$ [122]. A typical example of a heterogeneous graph is a Knowledge Graph (KG). KGs emphasize the contextual understanding of a graph, meaning that they are composed of interlinked sets of facts (relationships between entities in the real world) in a machine- and human-understandable format [54]. In addition to their categorization as directed/undirected and homogeneous/heterogeneous, graphs can exhibit another dimension by being either static or dynamic. In a dynamic graph, variations occur over time in terms of node and edge features or the overall topology of the graph. Figure 2.11 illustrates an example of a simple directed, heterogeneous, and static graph with three node types, three nodes, and three edges.

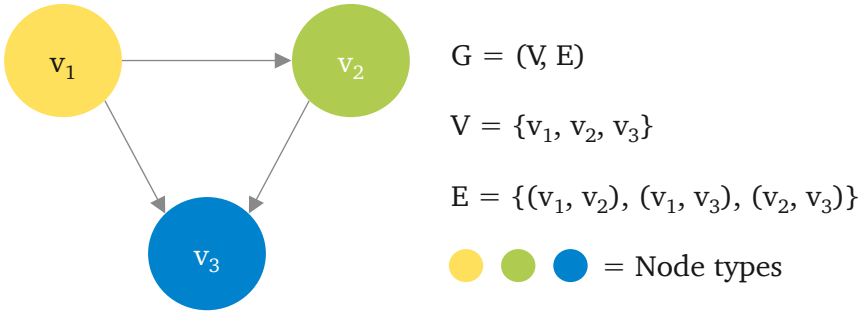


Figure 2.11.: Example of directed heterogeneous static graph

In graph-based data representation, a *data model* serves to systematically organize entities and relationships. For a KG, the fundamental components of a data model include entities, relationships, and attributes. Entities represent objects or concepts in the graph. A node with a unique identifier represents each entity and may have attributes that provide additional information about it. Entity classes are entities of the same type or category based on specific attributes or relationships. Relationships define connections between entities in a graph. They describe through edges how entities are related to each other and can have associated properties or attributes. The relationship types are usually described as verbs (predicates). Attributes provide information about entities or relationships and are usually stored as vectors in a graph associated with a node or edge. They can include properties such as names, dates, or numerical values.

Figure 2.12 illustrates a multi-stage process of a manufacturing plant containing two lines, four stations and five machines. Figure 2.13 contains an example of a data model for representing this manufacturing facility. The data model has three distinct entity classes (line, station, and machine) and three relationship types (indicating that a line has a station, a station has a machine, and a station transfers the semifinished products to another station). In this example, the relationships are directional, signifying the flow from line nodes to

station nodes, from station nodes to machine nodes, and from one station to the next station. Figure 2.14 illustrates a simple KG created based on this data model for line 1, stations 1 to 3, and machines 1 to 4. Each node has an attribute vector, with the attributes exemplified here consisting of one-hot-encoded vectors representing the line, station, and machine numbers.

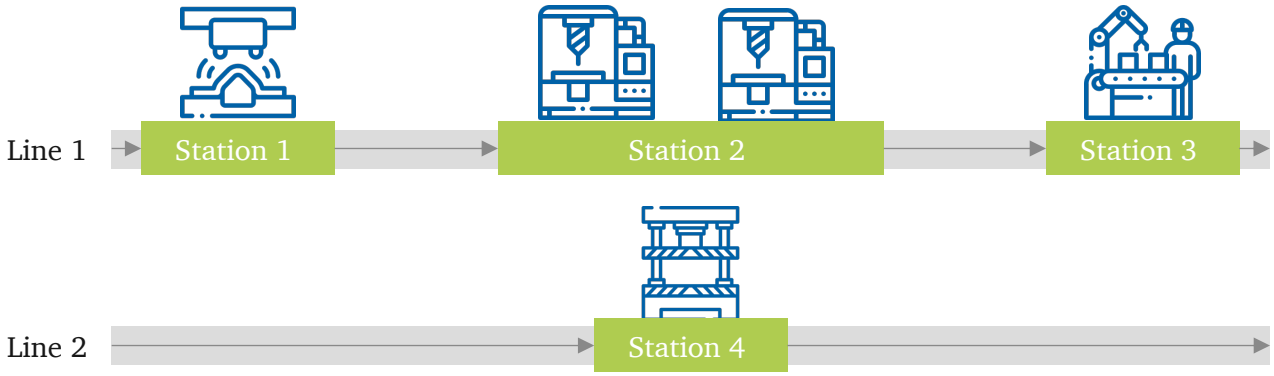


Figure 2.12.: Example of multi-stage process

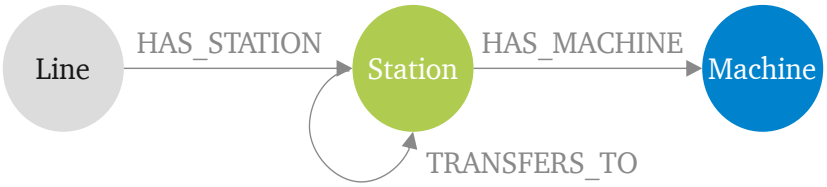


Figure 2.13.: Example of data model for the multi-stage process represented in Figure 2.12

The semantic nature of a graph is articulated through triples, encompassing a subject, a predicate (representing a relationship), and an object. Each triple encapsulates a statement about the relationship between two entities, namely the subject and the object. Within the context of Figure 2.13's data model, an illustrative triple takes the form of (Line [1,0], HAS_STATION, Station [1,0,0,0]). Triples are used to create KGs by defining the connections between nodes in a graph. Once the graph is defined, various graph learning tasks can be applied to either the individual graph or a collection of graphs constituting a graph dataset. Graph learning tasks, as categorized by Zhou et al. [126], typically falls into three main classes:

- Node-level: For performing node-level tasks, embeddings are created at the node-level. Embeddings in this case are dense vector representations capturing the structural properties of individual nodes. These embeddings are used for tasks such as node classification (categorizing nodes into predefined classes), node regression (predicting a continuous value for each node), and node clustering (grouping similar nodes into the same group).
- Edge-level: Edge-level tasks focus on how nodes are connected to each other for performing edge classification and link prediction (predict whether there is an edge between two nodes that are not currently present in the graph).
- Graph-level: For performing graph-level tasks, embeddings are created for the whole graph for performing graph classification (categorize whole graphs into predefined categories), graph regression (predict a continuous value for the whole graph), and graph matching (also called graph isomorphism, which deals with the problem of determining whether there is a one-to-one correspondence between the vertices of

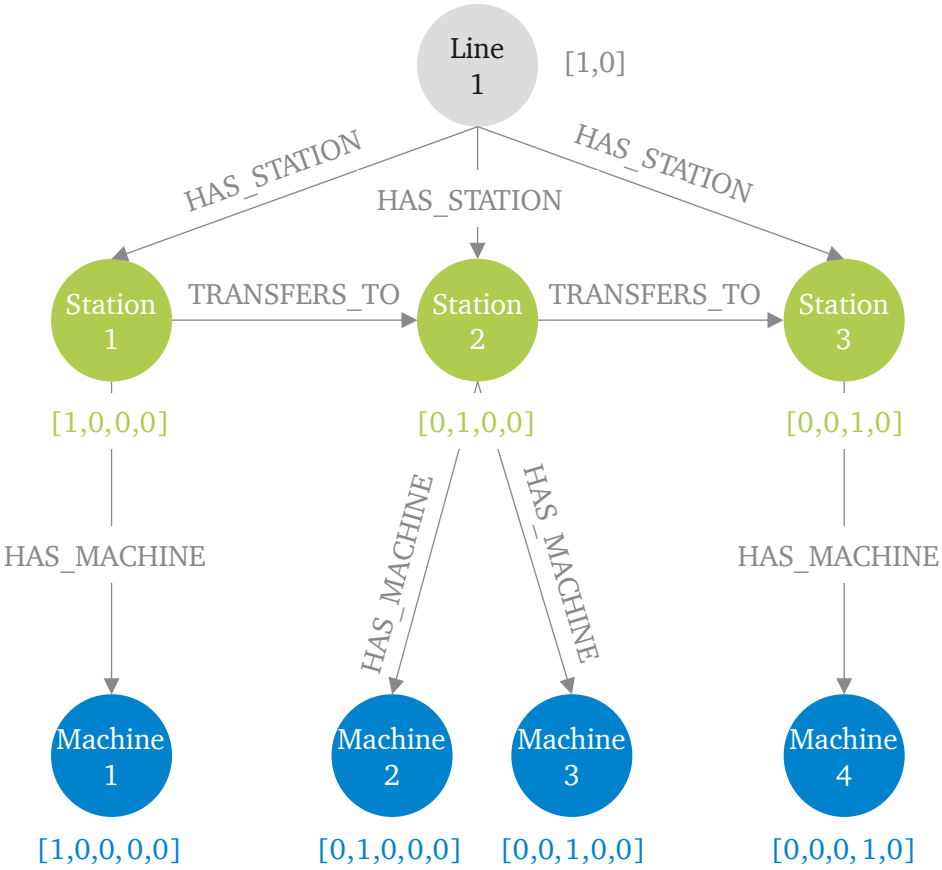


Figure 2.14.: Example of a KG for the multi-stage process represented in Figure 2.12

two graphs such that the adjacency relationships are preserved, it involves finding out if the structure of one graph can be exactly replicated in another graph).

The ML task on graph data covered in this thesis is supervised *graph classification*. In such an application, the dataset is composed of multiple different graphs associated with a graph-level label, and the objective is to make predictions specific to each graph. In this case, each graph is considered an independent and identically distributed (i.i.d.) data point associated with a label (informing the class it belongs to), and the goal is to use the labeled set of training samples to learn a mapping from graphs to its labels [47].

Researchers and practitioners use specialized techniques called Graph Neural Network (GNN)s to perform this type of task. GNNs are deep learning models designed to work directly with graph-structured data. Section 2.4.2 provides an overview of GNNs and a basic understanding of how they work.

2.4.2. Graph neural networks

This section draws heavily from the book titled *Graph representation learning* by Hamilton [47]. The GNN formalism is a general structure for defining deep neural networks on graph data. A GNN uses the so-called *message passing* framework for propagating information throughout the graph and generating representations of nodes and of the whole graph that depend on the graph structure. In each message-passing iteration in a GNN (also called as the different 'layers' of the GNN), a *hidden embedding* $h_u^{(k)}$ corresponding to each node $u \in V$ is updated according to information aggregated from u's graph neighborhood $N(u)$, as shown in Equations 2.60 and 2.61.

$$h_u^{(k+1)} = UPDATE^{(k)}(h_u^{(k)}, AGGREGATE^{(k)}(h_v^{(k)}, \forall v \in N(u))) \tag{2.60}$$

$$h_u^{(k+1)} = UPDATE^{(k)}(h_u^{(k)}, m_{N(u)}^{(k)}) \tag{2.61}$$

In these equations, $UPDATE$ and $AGREGATE$ are differentiable functions (i.e., mean or maximum value or a neural network) and $m_{N(u)}$ is the message being aggregated from u's neighborhood. The superscript (k) and $(k+1)$ indicate the state or iteration of message passing. The initial embeddings at $k = 0$ are the original input features of each node. At each iteration, the $AGGREGATE$ function receives as input the set of embeddings of the nodes in u's graph neighborhood $N(u)$ and generates a message $m_{N(u)}^{(k)}$ based on the aggregated neighborhood information. The $UPDATE$ function combines the message $m_{N(u)}^{(k)}$ with the node u's embedding $h_u^{(k)}$ to calculate the updated embedding $h_u^{(k+1)}$. At each layer of the GNN, or each iteration, every node embedding is updated to contain information about the feature of its immediate graph neighbors reached by a path of length one in the graph. After k iterations, every node embedding includes information from nodes up to k steps away, known as its k-hop neighborhood. The node embeddings contain two information types:

- Structural: Information about the graph topology, meaning the arrangement or layout of nodes and edges in the graph, which represents the relationships and connections between entities and concepts in a structured manner.
- Feature-based: Encoded information about the features of the node's neighboring nodes.

This structural awareness allows GNNs to effectively exploit the connections (such as connections between manufacturing processes, as illustrated in Figure 2.14) within the graph. Additionally, feature-based information enables GNNs to leverage both the local graph structure and the attributes associated with each node. This combination of structural and feature-based information processing enables GNNs to efficiently capture and utilize the interconnected nature of graph data, making them particularly well-suited for tasks involving connected processes. A basic GNN model for a graph-level prediction task is defined in Equation 2.62.

$$\mathbf{H}^{(k)} = \pi(\mathbf{A}\mathbf{H}^{k-1}\mathbf{W}_{neigh}^{(k)} + \mathbf{H}^{k-1}\mathbf{W}_{self}^{(k)} + \mathbf{B}^{(k)}) \tag{2.62}$$

In this equation, $\mathbf{H}^{(k)} \in \mathbb{R}^{|V| \times d}$ is the matrix of node representations at layer k, $\mathbf{A}$ is the graph adjacency matrix, $\mathbf{W}_{neigh}^{(k)}$ and $\mathbf{W}_{self}^{(k)}$ are trainable parameter matrices, π denotes an element-wise nonlinearity (e.g., ReLU), and $\mathbf{B}^{(k)}$ the bias term.

Graph Convolutional Network (GCN)s extend the principles of Convolutional Neural Network (CNN)s to accommodate graph data. While CNNs excel at handling various types of data represented in the form of arrays, effectively identifying and integrating visual patterns within these arrays (e.g., 1D arrays for signals, 2D arrays for images, and 3D arrays for videos) to create more comprehensive and meaningful representations [67], they are specifically tailored for data exhibiting a grid-like structure, known as regular Euclidean data. Consequently, they are ill-suited for graph data, which is usually defined in a non-Euclidean space [126]. The message passing function in GCNs is defined in Equation 2.63 [62], which uses a symmetric-normalized aggregation and a self-loop update to the node features.

$$h_u^{(k)} = \pi(W^{(k)} \Sigma_{v \in N(u) \cup u} \frac{h_v}{\sqrt{|N(u)||N(v)|}}) \tag{2.63}$$

Where 'symmetric-normalized aggregation' refers to how neighboring node features are aggregated in the message passing process. In Equation 2.63, the aggregation term $\sum_{v \in N(u) \cup u} \frac{h_v}{\sqrt{|N(u)||N(v)|}}$ involves summing

over neighboring nodes v of node u, including node u itself, and normalizing the sum by the square root of the product of the degrees of nodes u and v (denoted as $\sqrt{|N(u)||N(v)|}$). This normalization ensures that the aggregation is symmetric and invariant to the size of the neighborhood. 'Self-loop update' refers to updating the feature representation of each node by including its own feature in the aggregation process. In Equation 2.63, the term $\cup u$ includes node u itself in the set of neighboring nodes over which aggregation is performed. This self-loop update ensures that each node's feature representation incorporates information from the node itself in addition to its neighbors.

Graph convolutional layers aim to learn node representations by aggregating information from neighboring nodes. All graph convolutional layers have common aspects related to their operation mode:

- Node representations: Each node in the graph is associated with an initial feature vector representing its attributes or characteristics. The goal is to learn a meaningful representation for each node that captures information from its neighbors.
- Neighborhood aggregation: Graph convolutional layers aggregate information from a node's neighbors to update its representation. For each node, the layer considers its neighboring nodes and combines their information to create a new feature representation for the node itself.
- Learnable parameters: Graph convolutional layers have learnable parameters, which are weights associated with edges in the graph. These parameters are learned using backpropagation and optimization techniques during the training process.
- Convolutional operation: The aggregation of information from neighbors is performed using a convolutional operation. This operation is inspired by traditional convolutional layers used in image processing but adapted to graph-structured data. Instead of operating on a fixed grid like images, the convolution is applied to the graph structure.
- Activation function: After the aggregation, an activation function may be applied element-wise to introduce non-linearity to the learned representations.
- Pooling and downsampling: Some graph convolutional architectures incorporate pooling operations to downsample the graph, similar to how pooling is used in traditional convolutional neural networks for images. In GNNs, pooling typically involves aggregating information from neighboring nodes or subgraphs to create coarser graph representations, effectively reducing their size while preserving important structural information.
- Output: The final output of the graph convolutional layer is a set of updated node representations that can be used for downstream tasks such as graph classification.

Graph convolutional layers can be extended to perform graph-level classification tasks:

- Initialization: Like node-level tasks, each node in the graph is associated with an initial feature vector.
- Node aggregation: For each node, the graph convolutional layer aggregates information from its neighbors, updating the node's representation based on the features of its neighbors.
- Graph-level aggregation: After processing all nodes, the final step involves aggregating the node representations to obtain a graph-level representation. The graph-level aggregation is performed by the *readout function*. The readout function (also called *graph pooling*, since the goal is to pool together the node embeddings in order to learn an embedding of the entire graph) is used to generate graph-level representations based on node or edge representations. Typically, readout functions are simple and

non-adaptive functions (such as the mean, sum, minimum or maximum value of nodes' or edges' features) defined so the hypothesis space is permutation invariant [17].

- Output layer: The final layer's output is used for graph-level classification. This output is typically fed into a sigmoid layer to obtain probabilities for the classes in binary classification tasks.

A GNN is, therefore, an optimizable transformation on all attributes of the graph (nodes, edges, global context) that preserves graph structure and its symmetries [91]. By using a GNN to learn meaningful representations of heterogeneous graphs, we can consider the relationships within the data to solve classical ML tasks [45].

There are different variations of GNN layers, such as the ones explored in this thesis: Graph Convolutional Layer (GraphConv), GraphSAGE Layer (GraphSAGE), and Graph Isomorphism Network Layer (GIN). Each variant has different approaches to neighborhood aggregation or additional components, but the general idea is to capture and propagate information through the graph structure. Next, further details are provided for each layer type.

Graph Convolutional Layer (GraphConv) GraphConv is introduced in the paper "Semi-supervised classification with graph convolutional networks" from Kipf and Welling [63]. Mathematically, it is defined as shown in Equation 2.64 [30].

$$h_i^{(k+1)} = \pi\left(\sum_{j \in N(i)} \frac{1}{c_{ji}} h_j^{(k)} W^{(k)} + b^{(k)}\right) \tag{2.64}$$

Where:

- $h_i^{(k+1)}$ is the updated representation of node i at layer $k+1$,
- π is an activation function,
- $b^{(k)}$ is a learnable bias term,
- $N(i)$ is the set of neighbors of node i in the graph,
- c_{ji} is the product of the square root of node degrees, specifically $c_{ji} = \sqrt{\frac{|N(j)|}{|N(i)|}}$,
- $h_j^{(k)}$ is the representation (embedding) of the neighboring node j at layer k,
- $W^{(k)}$ is a learnable weight matrix.

The layer aggregates information from the neighboring nodes by considering the product of the square root of node degrees. This operation helps to normalize the information based on the degrees of neighboring nodes. After the aggregation step, the result is passed through an activation function π to introduce non-linearity to the model. The layer involves learnable parameters in the form of a weight matrix $W^{(k)}$ and a bias term $b^{(k)}$. These parameters are updated during the training process. GraphConv forms the basis for many subsequent advancements in graph convolutional architectures.

GraphSAGE Layer (GraphSAGE) GraphSAGE is introduced in the paper "Inductive representation learning on large graphs" from Hamilton, Ying, and Leskovec [46]. Equation **??** presents the mathematical definition of a GraphSAGE layer [31].

$$h_N^{(k+1)}(i) = \text{AGGREGATE}\left(\{h_j^{(k)}, \forall j \in N(i)\}\right) \tag{2.65}$$

$$h_i^{(k+1)} = \pi\left(W \cdot \text{CONCAT}(h_i^{(k)}, h_{N(i)}^{(k+1)})\right) \tag{2.66}$$

$$h_i^{(k+1)} = \text{NORM}(h_i^{(k+1)}) \tag{2.67}$$

Where:

- $AGGREGATE$ is the aggregation step. It collects information from the neighboring nodes $N(i)$ of node i in the graph. $AGGREGATE$ represents the aggregation function, which combines information from the neighboring nodes. The specific aggregation function depends on the GraphSAGE variant (e.g., 'sum', 'mean', 'pool', 'lstm').
- $CONCAT$ is the function that concatenates the current node's representation $h_i^{(k)}$ with the aggregated information $h_{N(i)}^{(k+1)}$ from its neighbors.
- W is a learnable weight matrix that is applied to the concatenated features. It transforms the concatenated feature vector into a new space.
- π is an activation function, often applied element-wise after the linear transformation with the weight matrix. Common choices include ReLU or another non-linear activation function.
- $NORM$ is a function that applies normalization to the updated node representation $h_i^{(k+1)}$. This step is optional and is often used to stabilize and standardize the features.
- $h_i^{(k)}$ and $h_i^{(k+1)}$ denote the node representations at layers k and $k+1$, respectively.

In summary, the GraphSAGE layer updates the representation of each node by aggregating information from its neighbors, concatenating the aggregated information with its current representation, applying a linear transformation (with a learnable weight matrix), and applying an activation function. The resulting representation is then normalized. This process is repeated for each node in the graph.

Graph Isomorphism Network Layer (GIN) GIN layers were first introduced in the paper "How powerful are graph neural networks?" from Xu et al. [117]. Its mathematical representation is shown in Equation 2.68 [32].

$$h_i^{(k+1)} = f_\Theta\left((1+\epsilon)h_i^{(k)} + \text{AGGREGATE}\left(\{h_j^{(k)}, j \in N(i)\}\right)\right) \tag{2.68}$$

Where:

- $h_i^{(k+1)}$ is the updated representation of node i at layer $k+1$,
- f_Θ is a neural network function with parameters Θ that operates on the aggregated information and the current node's representation,
- $(1+\epsilon)h_i^{(k)}$ is a self-loop term that scales the current node's representation. ϵ is a small positive constant,

- AGGREGATE $\left(\{h_j^{(k)}, j \in N(i)\}\right)$ is the aggregation of information from the neighboring nodes $N(i)$ of node i.

In summary, GIN incorporates a self-loop term, aggregating information from neighboring nodes, and applying an activation function followed by a neural network function. This design makes GIN suitable for tasks where the structure of the graph, rather than the node ordering, is crucial.

3. Existing approaches and required action

This chapter presents the existing approaches for predictive quality for multi-stage discrete manufacturing processes. The contemporary state of the research field is critically examined through a systematic literature review conducted in July 2023, subsequently published in the proceedings of the SEA4DQ Workshop 2023 at the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) [16].

This study aims to identify existing approaches and gaps in the current literature and articulate how this thesis can contribute to addressing the identified shortcomings. Section 3.1 outlines the methodology employed in the review process, details the research questions, and presents the key findings from the literature search. Section 3.2 provides an in-depth report on the main findings, offering comprehensive answers to the defined questions. Finally, Section 3.3 discusses the outcomes and formulates necessary actions for improving existing approaches.

3.1. Methodology, planning and conducting the review

The methodology used in the systematic literature review is based on the work of Kitchenham and Charters [64] and Xiao and Watson [116]. Kitchenham and Charters [64] propose a tripartite approach to execute a systematic literature review encompassing planning, conducting and reporting the study (see Figure 3.1).

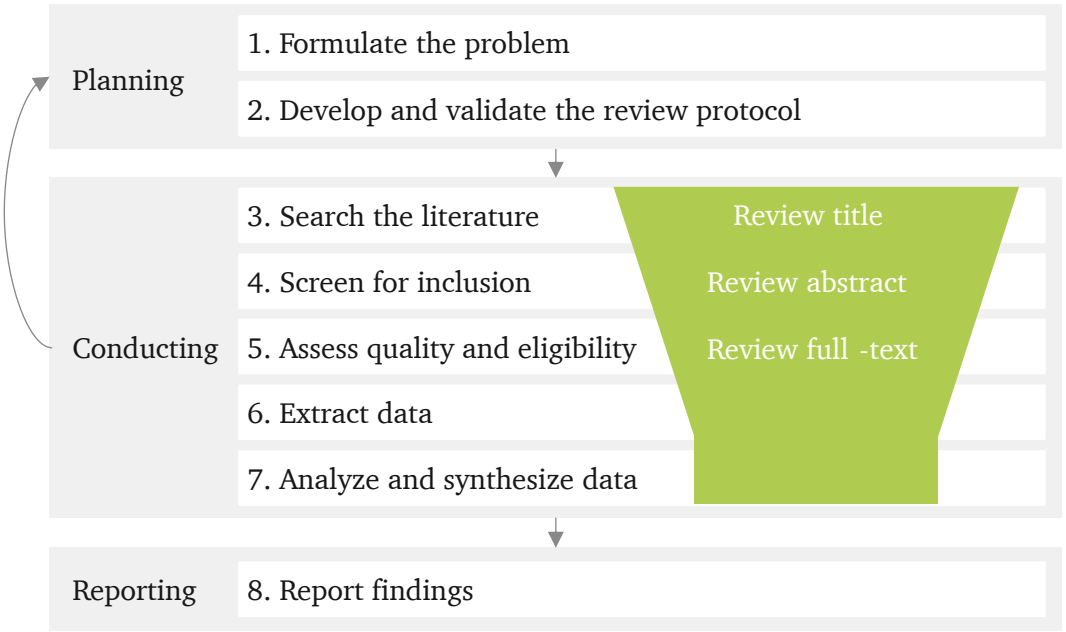


Figure 3.1.: Systematic literature review methodology according to Xiao and Watson [116]

The planning phase starts by identifying the need for a review, specifying the research questions, and

documenting each decision in a review protocol. This review protocol is a detailed description of the study's purpose, encompassing inclusion and exclusion criteria, search methodologies, criteria for evaluating the articles' quality, screening procedures, and methods for data extraction, synthesis, and reporting.

The second phase consists of the actual literature review execution. The stipulated search string narrows down the body of work and pinpoints potential studies. Articles are identified based on their titles and additional metadata using the defined search string. After the elimination of duplicates, abstracts of the articles are examined for suitability. Upon these appraisals, full-text articles are evaluated for eligibility under the criteria specified in the review protocol. Backwards and forward searches allow the identification of further articles to be included in the final set of studies. The ultimate set of articles can be subjected to data analysis and synthesis.

The third and final phase, termed reporting, involves the presentation of data extraction and analysis outcomes to address the research questions. Selection and interpretation bias mitigation has been achieved by engaging colleagues who offer an external perspective on the review process. This collaborative approach ensures a more comprehensive and balanced assessment, enhancing the objectivity and credibility of the review's findings.

In the present work, the purpose of the systematic literature review is to uncover developed practices for predictive quality in multi-stage manufacturing processes, highlight existing gaps, and elucidate how this thesis can aid in bridging these gaps and pushing the frontiers of knowledge in the field. Two Research Questions (RQs) have been formulated to guide the data extraction and achieve the defined purpose:

- RQ1: How is predictive quality currently performed for multi-stage discrete manufacturing processes?
- RQ2: What are the current challenges and limitations of predictive quality for multi-stage discrete manufacturing processes?

The search string is defined as *(quality) AND (predict* OR forecast* OR classif*) AND (multi-stage OR multi-step OR production line) AND (discrete manufactu*)*. The terms enclosed within the initial two parentheses encompass the predictive quality aspect. The subsequent pairs of parentheses encapsulate both the multi-stage and discrete manufacturing dimensions. The search string has been adapted to align with the search engines of the selected digital libraries, namely ACM [2], IEEE Xplore [52], and ScienceDirect [95]. The search is bounded to the last five years (from 2018 to 2023) to ensure the research aligns with the most up-to-date relevant findings and addresses the identified gaps that have emerged in the literature. The resulting search outputs are as follows: ACM Digital Library yields 54 results, IEEE Xplore presents 27 results, and ScienceDirect provides 24 results. Altogether, a collection of 105 potential titles is analyzed.

During the screening process, articles were first assessed based on their titles and abstracts and excluded if they were unrelated to manufacturing or predictive quality in discrete manufacturing. In this step, the keywords and phrases that indicate a focus on manufacturing processes, industries, or technologies, such as terms like 'manufacturing', 'production', 'industry', 'factory', or specific manufacturing sectors (e.g., automotive, electronics), and a focus on 'predictive quality', such as 'quality prediction', 'quality control', 'defect prediction', or similar terms were searched. Full-text articles were also subject to exclusion criteria. Those were excluded if their primary focus did not align with predictive quality (e.g., anomaly detection, optimization, planning, and scheduling) or fell below a certain length threshold: Having fewer than five pages in double-column format or six pages in single-column format. Articles that did not provide a concrete predictive quality application or discussed broader themes like general IoT frameworks, knowledge management, process monitoring, or theoretical models were also deemed unsuitable for inclusion.

To illustrate this progression, Figure 3.2 graphically presents the sequential search results, following the framework proposed by Xiao and Watson [116]. In the screening phase, three duplications were removed, and all abstracts were evaluated. Subsequent exclusion based on the defined criteria, as delineated in Figure 3.2,

led to a refined pool of 49 papers, all of which underwent thorough full-text assessment. Within this set, 24 papers were deemed ineligible, each exclusion rationale elucidated within Figure 3.2. 19 supplementary articles emerged via iterative searches encompassing both forward and backward searches. The inclusion step culminated in identifying 44 pertinent articles selected for comprehensive data extraction and analysis.

Besides descriptive data about the publications, such as year and country of publication, to address RQ1, additional questions have been defined to guide the data extraction process and answer RQ2:

1. What are the mentioned challenges?
2. What are the approaches' limitations?
3. Do the authors propose any future work?
4. How is the multi-stage aspect taken into consideration?
5. Which metrics are used to evaluate the approach?
6. How successful is the presented approach?

The final list of papers can be found in Table 3.1. Section 3.2 reports on the findings and provides answers to the RQs.

Table 3.1.: Final list of publications analyzed in the systematic literature review

Number	Title	Year	DOI	Reference
1	Surface quality evaluation based on roughness prediction model	2018	10.1145/3148453.3306271	[68]
2	Wafer map defect pattern classification and image retrieval using CNN	2018	10.1109/TSM.2018.2795466	[79]
3	Multitask learning for virtual metrology in semiconductor manufacturing	2018	10.1016/j.cie.2018.06.024	[83]
4	Analytical approach for fault diagnosis and quality control	2018	10.1016/j.procir.2018.03.024	[51]
5	Semantic weldability prediction with RSW quality dataset	2018	10.1016/j.aei.2018.05.006	[61]
6	Joint modeling of classification and regression for faulty wafer detection	2018	10.1007/s10845-018-1447-2	[58]
7	A novel visual fault detection and classification system for semiconductor manufacturing	2019	10.1109/ETFA.2019.8869311	[92]
8	What are the most informative data for virtual metrology?	2019	10.1109/COASE.2019.8842942	[73]
9	How IoT and computer vision could improve the casting quality	2019	10.1145/3365871.3365878	[84]
10	A Markovian approach for time series prediction for quality control	2019	10.1016/j.ifacol.2019.11.480	[90]
11	A recurrent neural network architecture for failure prediction	2019	10.1016/j.promfg.2019.06.205	[76]

Table 3.1 (continued)

Number	Title	Year	DOI	Reference
12	CNN for wafer surface defect classification and detection	2019	10.1109/TSM.2019.2902657	[23]
13	Deep learning architecture for virtual metrology modeling	2019	10.1109/TSM.2018.2849206	[73]
14	Data-driven fault diagnosis in end-of-line testing	2019	10.1109/DSAA.2019.00064	[49]
15	Interpretable multi-task learning for product quality prediction	2019	10.1109/ICDE.2019.00207	[118]
16	Multistage quality control using machine learning	2019	10.1109/ACCESS.2019.2923405	[86]
17	An intelligent metrology informatics system based on neural networks	2019	10.1016/j.procir.2019.04.148	[82]
18	Exploiting domain knowledge for classification tasks	2020	10.14778/3415478.3415549	[50]
19	Convolutional encoder-decoder architecture for quality prediction	2020	10.1145/3340531.3412007	[24]
20	Predicting quality of automated welding with machine learning	2020	10.1145/3340531.3412737	[126]
21	Quality control in injection molding based on norm-optimal	2020	10.1016/j.ifacol.2020.12.2777	[101]
22	Convolutional neural network based Gaussian process regression	2020	10.1016/j.conengprac.2019.104262	[114]
23	Adversarial bidirectional LSTM-based QTD framework	2020	10.1007/s10845-019-01530-8	[71]
24	Real-time identification of successful snap-fit assemblies	2020	10.1109/TASE.2019.2932834	[35]
25	Predictive model-based quality inspection using Machine Learning	2020	10.1016/j.aei.2020.101101	[93]
26	Virtual in-line inspection for function verification	2020	10.1016/j.procir.2020.03.126	[108]
27	Welding quality analysis and prediction based on deep learning	2021	10.1109/WCMEIM54377.2021.00045	[70]
28	Noncontact reflow oven thermal profile prediction	2021	10.1109/TCPMT.2021.3120310	[69]
29	Gaussian mixture model clustering ensemble regressor	2021	10.1109/ACCESS.2021.3055433	[55]
30	Data-based approach for final product quality inspection	2021	10.1109/CDC45484.2021.9683231	[60]
31	Smart sampling scheme C2O utilizing virtual metrology	2021	10.1109/ACCESS.2021.3103235	[104]
32	Mechanical fault detection for induction motors	2021	10.1109/IECON48115.2021.9589189	[28]
33	Quality prediction in semiconductors using unlabeled data	2021	10.1145/3460824.3460855	[85]

Table 3.1 (continued)

Number	Title	Year	DOI	Reference
34	Automatic visual inspection of turbo vanes	2021	10.1145/3441233.3441241	[27]
35	Machine learning for in-circuit testing of PCB assembly	2021	10.1145/3508259.3508291	[53]
36	Sequential inspection procedure for fault detection	2021	10.3390/s21227524	[77]
37	Battery production design using multi-output machine learning	2021	10.1016/j.ensm.2021.03.002	[107]
38	Contrastive decoder generator for few-shot learning	2022	10.1109/TII.2022.3190554	[119]
39	Path enhanced bidirectional graph attention network	2022	10.1109/TII.2021.3076803	[120]
40	Fault detection in multi-stage manufacturing	2022	10.1109/ICCAD55197.2022.9853909	[59]
41	Event-triggered feedforward predictive control	2022	10.1109/TII.2021.3082187	[41]
42	Quality prediction in a smart factory	2022	10.1145/3548785.3548796	[8]
43	Wafer-level characteristic variation modeling	2023	10.1145/3566097.3567915	[78]
44	Core industry manufacturing process of electronics assembly	2023	10.1145/3529098	[22]

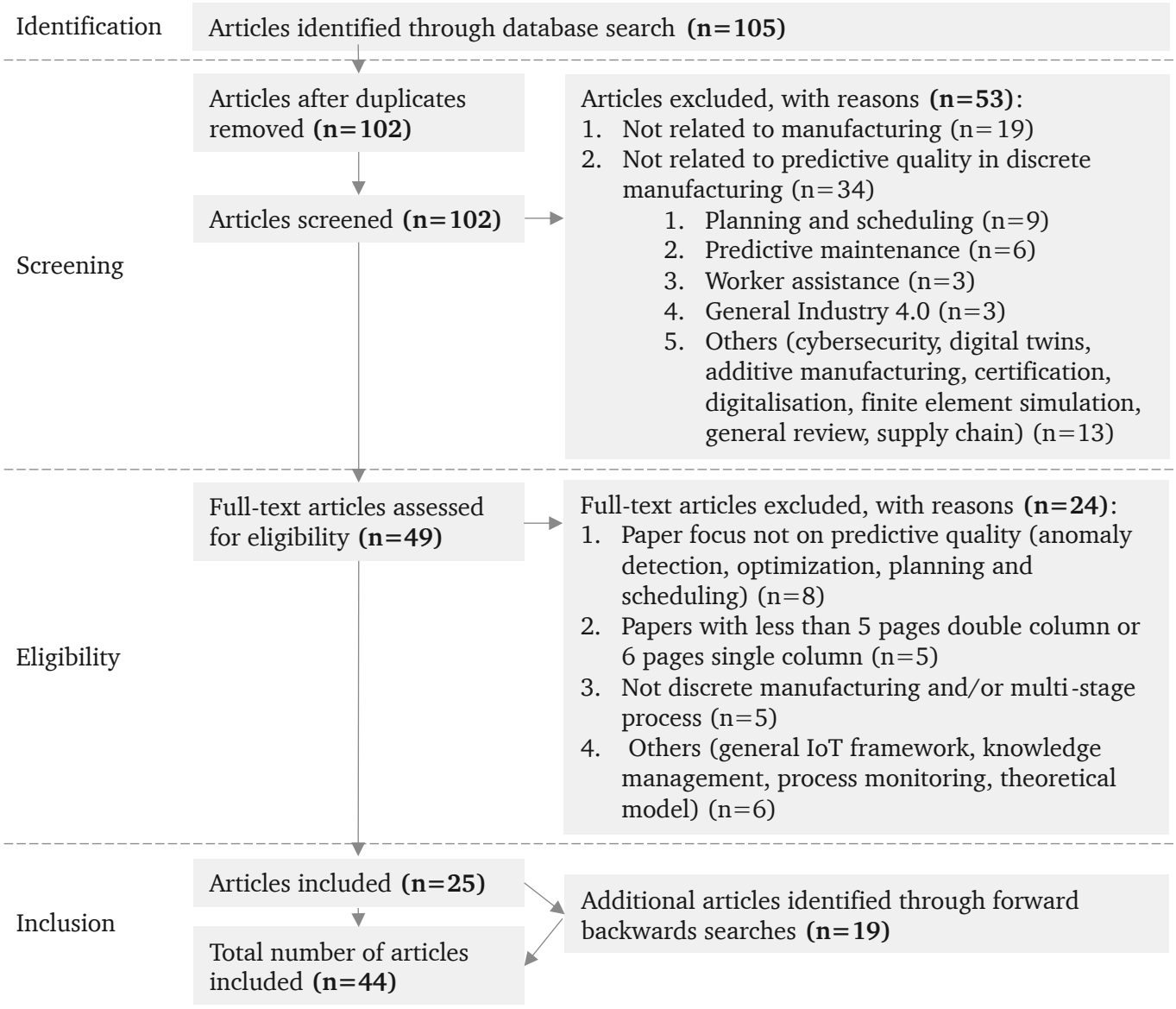


Figure 3.2.: Results of the systematic literature review

3.2. Reporting and discussion

RQ1 Figure 3.3 presents the overview of publication trends over the past five years.

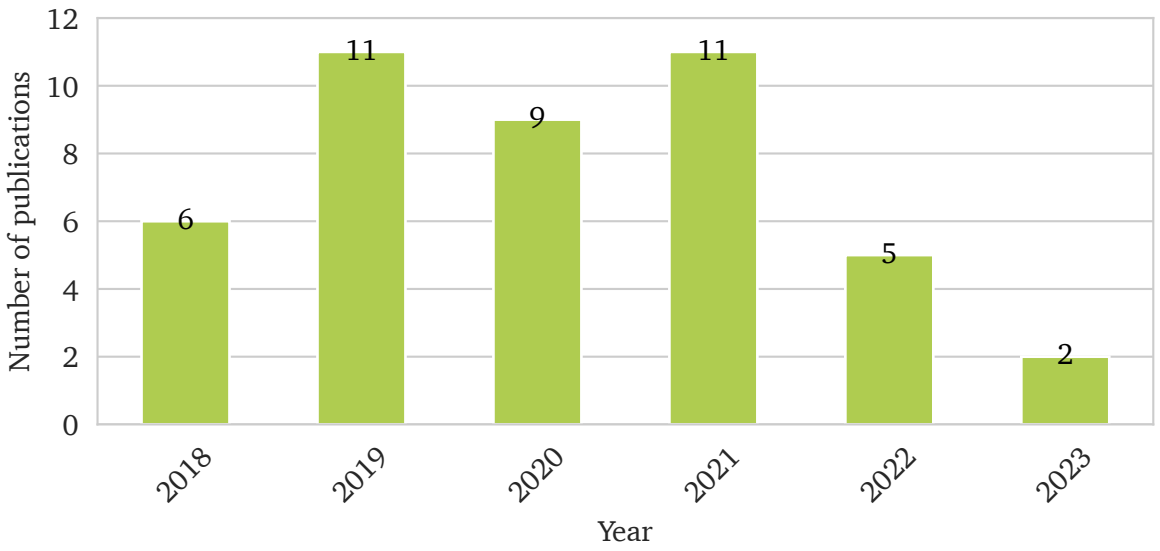


Figure 3.3.: Publications per year

Notably, there is an observed decrease of approximately 55.0% in publications between 2021 and 2022. This decline can be attributed to the outcomes of both backward and forward searches, which reference publications from previous years. Additionally, it is essential to consider that predictive quality studies developed in 2022 and 2023 might still be in the process of being published. However, a broader literature review on predictive quality in manufacturing and not specific to data quality and multi-stage discrete manufacturing processes conducted over an extended timeframe reveals an upward trajectory in the number of publications [103].

Moving to Figure 3.4, the distribution of papers is categorized based on the primary location of the first author's research institution or company. Particularly striking is the prevalence of articles originating from Germany (25.0%) and China (approximately 18.2%). These countries are known for their robust manufacturing sectors, where quality control and predictive techniques hold significant importance [112]. This emphasis likely emerges from their commitment to maintaining elevated product standards and remaining competitive in global markets.

Figure 3.5 delves into the origins of case studies featured in the publications, highlighting a significant focus on semiconductors and semiconductor equipment (34%, predominantly related to silicon wafer production) and the automotive industry (approximately 18.2%, primarily focused on assembly processes). This prominence is understandable considering the dynamic landscape of the electronics sector, which continuously pushes advancements in silicon wafer manufacturing techniques, materials, and technologies. Simultaneously, the stringent safety and quality standards imposed on automobiles motivate researchers to concentrate on assembly processes to ensure that vehicles meet regulatory mandates and fulfil consumer expectations. Figure 3.5 also points out possibilities for future work in less explored industries, such as machinery, household durables, health care equipment and supplied or additional industries not found in the existing studies.

The Sankey diagram in Figure 3.6 presents the number of diverse data types (categorized as tabular, time series, image, and textual), the developed tasks (classification, regression, clustering or model-based optimization) and the use of ML or deep learning and alternative methods identified within the articles. A

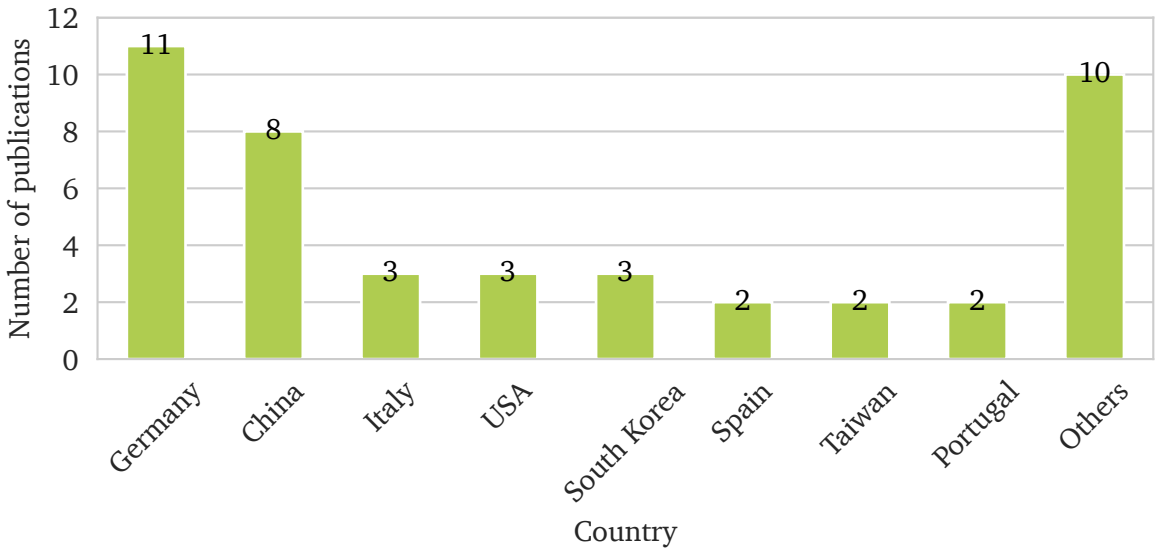


Figure 3.4.: Publications per country

prominent portion (43.2%) employs tabular data, characterized by its structured arrangement resembling spreadsheets. This data format encompasses rows representing individual entries and columns containing specific attributes or fields. Time series data, constituting a sequence of time-stamped data points, constitutes 22.7% of the publications. Eight papers (18.2%) showcase a fusion of tabular and time series data. Image data, central for computer vision systems, accounts for 11.36% of the case studies. Notably, textual data was present in only one paper, in which features from textual alarm events were harnessed for product quality classification [60]. Additionally, a single article employs all three data modalities in a quality control system for ceramic circuit boards [22]. The application of tabular data primarily gravitates towards classification tasks (66.7% for binary classification between conforming and non-conforming products). Meanwhile, time series data predominantly fuels regression tasks for estimating specific quality attributes. Clustering was initially used in some cases [78, 23] to prepare the data and identify sub-sets for which classification or regression models were trained. 47.7% of the publications used ML methods in their approaches, with the top three algorithms being RF, SVM and kNN. Deep learning methodologies hold a share of 22.7%, with CNNs and Long Short-Term Memory networks being the prevailing architectures. 20.4% implemented both ML and deep learning algorithms to develop predictive quality solutions. One paper [61] combined semantic rules with decision trees for estimating a quality parameter in a resistance spot welding process. Only three publications presented model-based approaches for quality estimation and optimization [101, 41, 77]. The Sankey diagram identifies the prevalent data types as tabular, time series data, and analytics tasks as classification and regression. By visualizing the distribution of flows between different categories, we can discern patterns in methodological choices, particularly the broader usage of ML techniques over deep learning. This preference could stem from the inherent nature of deep learning algorithms, which often demand extensive training datasets to recognize patterns within the data effectively. Additionally, it is worth noting that deep learning models tend to offer less interpretability in comparison to conventional ML models like tree-based algorithms. The insights derived from the Sankey diagram offer several potential directions for future research. These include investigating the application of ML and deep learning techniques to less common data types, improving the efficacy of deep learning methods for prevalent data types and tasks, and exploring multi-modal approaches for classification and regression tasks.

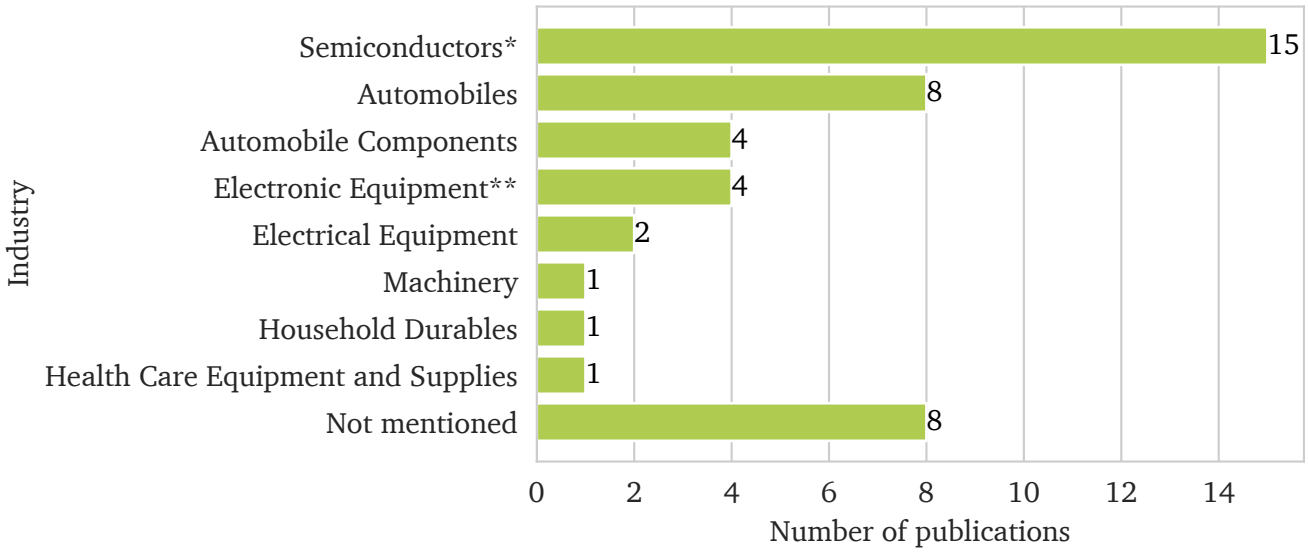


Figure 3.5.: Publications per industry. *Semiconductors and semiconductor equipment, **Electronic equipment, instruments and components

RQ2 Challenges: To answer the second RQ, the challenges, typically described in the motivation of each study, were systematically compiled and grouped into four primary clusters. The first cluster encompasses challenges related to *data challenges and quality control* [84, 24, 126, 70, 69, 55, 60, 27]. The prevalent obstacle reported involves the acquisition of ample datasets for effectively training ML (mainly deep learning) models and challenges with missing or insufficient data. Also, in this cluster, hurdles dealing with heterogeneous data sources and formats, high dimensional input parameter spaces and process variations are often mentioned.

The second cluster identifies obstacles related to *quality control techniques and methods* [90, 50, 101, 28, 120]. The most cited challenge is the need for alternative approaches to quality control methods that are costly and time-consuming. The authors also refer to manual quality controls and manual analysis of sensory data recorded throughout the manufacturing process conducted by scarce domain experts. The domain knowledge from quality engineers is usually hard to be formalized or persist consistently for future reference [76].

The third cluster is dedicated to challenges related to the *production process* [73, 51, 118, 61, 58, 114, 71, 86, 77, 35, 93, 108, 82, 107]. The most significant identified hurdle is the complexity of quality control, especially end-of-line quality control, in multi-stage manufacturing processes due to the interdependencies between stations and the propagation of errors and variation accumulation from multiple stages. The growing complexity and personalization of products only add to product inspection and quality control challenges. This increasing complexity is only aggravated due to high-quality standards and increasing costs for the metrology of all products.

The fourth and last cluster contains publications identifying *challenges in specific manufacturing areas* [92, 73, 49, 104, 41, 53]. Most challenges refer to specific product characteristics, such as mounting electronic components on Printed Circuit Board (PCB) and verifying their quality before scrap products are produced. The authors also mention challenges in the quality control process of assembly lines, particularly in the automotive sector. Several sources present challenges fitting to more than one cluster. Specifically, sources [73, 51, 118, 86] intersect clusters 1 and 3. Similarly, sources [77, 35, 93, 108] fit into clusters 3 and 4.

Limitations: The top five limitations of the presented approaches have been extracted to address the second inquiry. The first and most frequently encountered limitation pertains to the *scarcity of available samples (and datasets)* for evaluating the proposed approaches [72, 28, 59, 8, 107]. Limitations concerning the number of labelled examples for supervised learning and the absence of high-quality data for enhancing the robustness and generalizability of ML models are reported by the authors.

The second most commonly encountered limitation is also tied to the available data for training and testing. The *low or insufficient performance of the models* and their need for enhancement are highlighted [70, 104, 120, 58, 108]. Due to the low values of evaluation metrics, the deployment of many models into production is compromised.

The third limitation is associated with the *efforts required for feature engineering* and the manual efforts involved in devising relevant features to train the ML models [126, 61, 50, 55, 60]. Some authors [68, 76] indicate the *lack of consideration of further process factors* in their approaches, although these factors are acknowledged to be pertinent for evaluating product quality. Approaches rooted in computer vision systems that analyze image data for product quality classification emphasize a *strong dependency on camera settings* (including positioning and lighting) to achieve satisfactory outcomes [84, 92, 27]. Furthermore, these approaches commonly omit the multi-stage aspect of the production process and are not suited for functionality testing [92, 84].

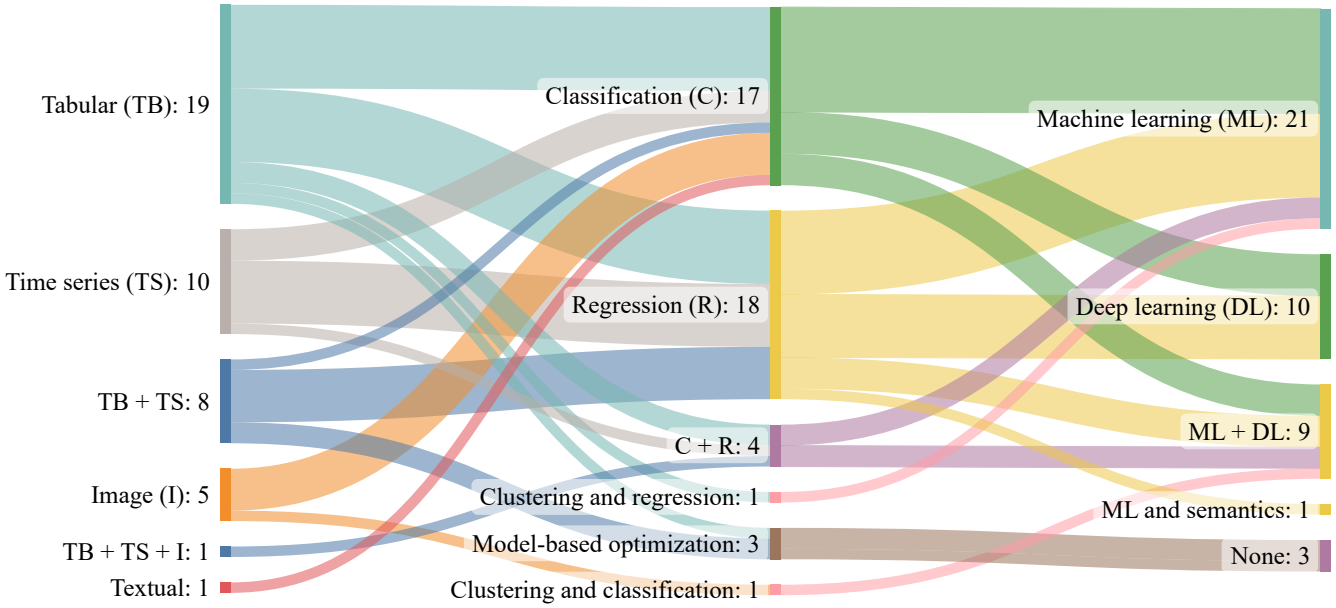


Figure 3.6.: Sankey diagram - links between data types, data analytics tasks and the use of machine or deep learning methods

Future work: When provided, future work suggests addressing the acknowledged limitations: acquiring more reliable data, improving models' performance, trying out different feature engineering techniques, and considering process factors and the multi-stage aspect of the manufacturing processes. Particularly on the last aspect, Kang [58] propose changing the modelling approach to incorporate related tasks in the manufacturing process and their complex relationships to improve predictive performance. However, It is unclear how the authors intend to do so. Wu et al. [114] mention the potential use of graph data formats to

model the multi-stage data to provide more intrinsic characteristics to deep learning models and improve model performance. Zhang et al. [119] is notably the only publication that uses graph representations of the data (homogeneous graphs) to train deep learning models for the predictive quality task. In their future work, the authors emphasize the need to try heterogeneous directed graphs to achieve more expressive data and better model outputs, which aligns with this thesis's proposed approach. Looking holistically, the upcoming directions for future research are strongly based on improving the quality, expressiveness and amount of data used to train and assess ML and deep learning models. This data augmentation goes beyond already existing sources, extending its reach to encompass additional data origins such as from retrofitted sensors. As the predictive quality landscape evolves, there is an inclination towards harnessing new data representation methodologies. Graph representations (such as KGs), in particular, have emerged as a potent ally, enabling the modelling of complex relationships and interdependencies intrinsic to multi-stage discrete manufacturing systems.

3.3. Critical evaluation and required action

The findings outline various aspects of the research landscape, including publication trends, geographic distribution, case study origins, data types, methods, and algorithms used in predictive quality studies. Case studies primarily focus on industries like semiconductors and semiconductor equipment and the automotive sector, characterized by complex manufacturing processes and stringent quality standards. Regarding data types and methods, the Sankey diagram in Figure 3.6 illustrates the diversity of data types used in predictive quality studies. It provides insights into the use of tabular, time series, image, and textual data, as well as classification, regression, clustering, and model-based optimization tasks. ML algorithms are preferred over model-based approaches to address predictive quality challenges. This assessment makes it possible to identify baseline approaches for comparing the method proposed in the current work. These baseline approaches are divided between ML and deep learning and summarized in Table 3.2.

Table 3.2.: Baseline approaches

Approach	**Algorithm**
ML	Logistic Regression (LR)
	Random Forest (RF)
	Support Vector Machine (SVM)
	k-Nearest Neighbor (kNN)
	eXtreme Gradient Boosting (XGBoost)
Deep learning	Multilayer Perceptron (MLP)
	Convolutional Neural Network (CNN)
	Long Short-Term Memory (LSTM)

The search results outline the limitations and challenges researchers face in the domain of predictive quality for multi-stage discrete manufacturing processes. The identified limitations and challenges are summarized into five categories:

1. Scarcity of available data: A significant limitation is the need for more samples and datasets for training and evaluating predictive quality models. This limitation impacts model robustness, generalizability, and overall performance.

2. Model performance: Many studies report low or insufficient model performance, indicating the need for improvement. This limitation hampers the successful deployment of models into real-world production environments.

3. Feature engineering efforts: Authors often encounter challenges in feature engineering and the manual steps required to devise relevant features for ML models. Certain aspects of the manufacturing process, such as their interdependencies, are not adequately considered, leading to limitations in feature representation and data expressiveness.

4. Complexity of multi-stage processes: The complexity of quality control in multi-stage manufacturing processes is highlighted as a limitation. The interdependencies between stages, error propagation, and accumulation issues pose challenges for accurate quality assessment.

5. Specific manufacturing challenges: Some studies mention limitations related to particular manufacturing areas, such as product characteristics and quality control processes for certain products like electronic components on PCBs.

In response to the challenges and limitations identified in this chapter and building upon the exploration of emerging technologies for graph representation learning in Section 2.4, this thesis introduces an innovative data modeling and analysis approach. This approach uses a graph representation of the data and leverages the capabilities of GNN to learn patterns. The primary objective is to address the limitations identified in categories two, three, and four:

- Model performance: Diverse data sources, such as tabular and time series data, can be integrated into a unified representation as a heterogeneous graph. This integration of heterogeneous data sources and the representation of their relationships enhances the data expressiveness, potentially improving the model's robustness and generalizability. GNN excels in capturing complex relationships and dependencies within a graph, potentially outperforming baseline algorithms by modeling complex interdependencies and deviations in multi-stage manufacturing processes.

- Feature engineering efforts: The manual effort required for feature engineering is alleviated through a GNN-based approach, as these neural network architectures inherently capture and learn relevant features from the network structure, including interdependencies between multi-stage manufacturing processes.

- Complexity of multi-stage processes: GNN models can learn to identify patterns and relationships that impact the final product quality, considering the interconnected nature of the process.

In summary, the justification for adopting a novel data modeling approach based on heterogeneous graphs and GNNs lies in its capacity to address the identified limitations and challenges. This approach leverages the graph structure to integrate diverse data sources, capture complex relationships, account for multi-stage complexities, and adapt to specific manufacturing domains. Addressing the limitations from categories one and five poses challenges that do not align with the goals and scope of this thesis. For category one, the scarcity of available data is a fundamental constraint and case-dependent that extends beyond the control or influence of the researcher. As for category five, specific manufacturing challenges are often intrinsic to particular industries or product types. These challenges may require domain-specific expertise and interventions that extend beyond the scope of a generalized predictive quality method. Addressing these challenges necessitates investigating specific cases, exploring the respective manufacturing domains, and tailoring solutions to meet their unique requirements, which may divert the focus from the broader objectives of this thesis.

4. Research design

Engineering research is motivated by solving a practical and relevant problem. The initial investigation and understanding of this defined problem motivate a research question. This question, in turn, delineates the research project—an organized series of activities designed to lead to a conclusive result or answer. This outcome contributes to resolving the underlying practical problem. Booth et al. [12] call this iterative process the research flow, depicted in Figure 4.1.

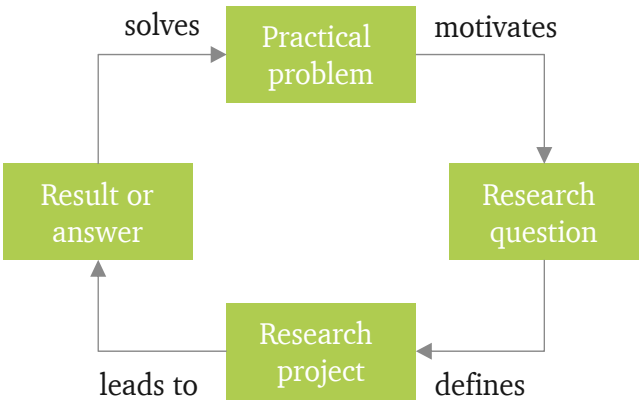


Figure 4.1.: Research flow according to Booth et al. [12]

The practical problem the field of predictive quality aims to address is how to develop and implement predictive models to anticipate and assess the quality of a product, process, or system, enabling proactive measures and improvements before quality issues arise and defective products are manufactured, thus reducing quality control costs and improving overall quality [103]. Within this broader context, this thesis focuses on the problem of implementing predictive quality for multi-stage discrete manufacturing processes for replacing or reducing end-of-line quality control. The systematic review conducted in Chapter 3 evidenced the gaps guiding this research: Finding ML models that perform better than existing approaches and reducing feature engineering efforts while capturing the complexity and interrelationships of multi-stage processes.

The research process that orients this thesis is shown in the flow chart in Figure 4.2, adapted from the book *Research methodology: Methods and techniques* from Kothari [65]. The research process starts with defining the research problem and reviewing existing literature. The review of concepts and theories and previous research findings from existing approaches were presented in Chapters 2 and 3, respectively. The third step is formulating a working hypothesis regarding the research problem. The hypothesis should be specific and limited to the research scope since it has to be tested. It also indicates the type of data required and the data analysis methods to be used. After that, in steps four to six, data is collected and analyzed so conclusions can be drawn on how to solve the problem and validate the proposed solution by accepting or rejecting the working hypothesis. Finally, the results obtained are interpreted and reported in the last step. The research process is interactive; the feed-forward arrows indicate the information from a specific step that provides information for evaluating the activities in another step. The feedback arrows indicate the direction of the information flow that will help control the transmitted task.

Feed-forward arrows:

- (IV) Design research → (VI) Analyze data: The research design establishes the parameters for the data analysis.
- (I) Define research problem → (VII) Interpret and report: The research problem provides the necessary criteria to evaluate if the results help solve the identified problem.

Feedback arrows:

- (V) Collect data ← (VI) Analyze data: The data analysis helps understand if the collected data is appropriate for achieving the desired objectives and accepting or rejecting the hypotheses.
- (VI) Analyze data ← (VII) Interpret and report: The interpretation and reporting of results help control and refine the data analysis.
- (I) Define research problem ← (VII) Interpret and report: Insights from the interpretation of the results help define and delimit the scope of the research problem.

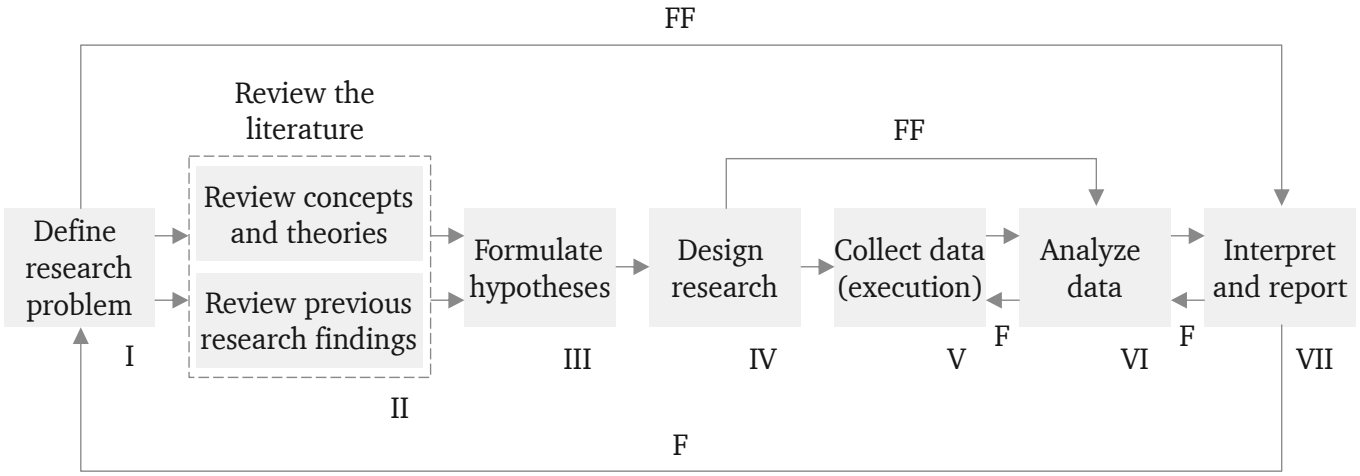


Figure 4.2.: Research methodology according to Kothari [65]. F = feedback, FF = feed-forward

Based on the defined research problem, the identified research gaps and recent developments of ML on graphs, the objective is to propose and evaluate a novel data modeling approach for predictive quality in multi-stage discrete manufacturing processes using heterogeneous graph representations and GNN. This approach is compared with currently used methods determined in Chapter 3 and suited for tabular and time series data, more precisely LR, RF, SVM, kNN, XGBoost, and MLP. The research question underlying this research objective is:

Research question:

How does the performance of the proposed novel approach, utilizing heterogeneous graph representations and GNN, compare to the identified baseline ML algorithms?

Based on this research question, the following hypothesis is derived:

Hypothesis:

The proposed approach outperforms the baseline ML algorithms.

The data collection (V), data analysis (VI), and results interpretation and reporting (VII) are conducted twice for two case studies. For each case study, the baseline ML algorithms are trained and tested with the same data used for creating the heterogeneous graphs and contrasted with the graph representation of the process data. In both cases the ML task is a binary classification, the objective is to use process data to classify the product as either belonging to the positive class (non-quality conform products, label equals to '1') or negative class (quality-conform products, label equals to '0').

The CRoss-Industry Standard Process model for the development of Machine Learning applications with Quality assurance methodology (CRISP-ML(Q)) [102], represented in Figure 4.3, is chosen as the research design (IV) and used to structure the case studies and develop the predictive quality applications.

Figure 4.3.: (a) CRISP-ML(Q) according to [102]

The CRISP-ML(Q) is composed of six phases:

1. Business and data understanding: The first phase focuses on understanding the project objectives and requirements from a business and data perspective and how the collected data can help achieve these objectives. It involves identifying goals, objectives, and requirements and converting them into data mining problems. In this phase, data collection, exploration, and initial analysis occur. It involves gathering data, describing its characteristics, exploring its quality, and verifying whether the available data is relevant to the project goals.

2. Data preparation: Data preparation involves data preprocessing and cleaning to make it suitable for analysis. The tasks in this phase include data cleaning, integration, transformation, feature engineering, and feature selection. The goal is to ensure that the data is in a suitable format for modeling.

3. Modeling: In the modeling phase, various modeling techniques are selected and applied to the prepared data. This thesis's modeling involves building and evaluating models using the proposed approach and the baseline ML algorithms.

4. Evaluation: The models generated in the previous phase are evaluated to determine their effectiveness and suitability for the business problem at hand. Evaluation metrics are used to assess model performance, and the best-performing model(s) are selected for deployment. The proposed approach will be contrasted with the selected baseline algorithms in this phase.

5. Deployment: This phase involves deploying the selected model(s) into a production environment. It includes integrating the model into existing systems, monitoring its performance, and providing ongoing support as needed.

6. Monitoring and maintenance: After deployment, the model is continuously monitored to ensure its performance remains optimal. This phase involves tracking model performance, detecting concept drift, retraining the model as needed, and addressing any issues that arise during production.

These phases are iterative, meaning that it is expected to revisit previous phases as new insights are gained or as the project requirements evolve. In the context of this thesis, a particular focus is given to the phases 1, 2, 3 and 4, namely business and data understanding, data preparation, modeling and evaluation. Next, a short summary of each case study is provided.

Case study 1: Bosch production line performance The first case study is based on an open-source dataset from the company Bosch, the 'Bosch production line performance'. The dataset contains data recordings of products assembled in multiple stations in four assembly lines and the labels indicating whether the product passed or failed the end-of-line quality control. The goal is to train a classifier to predict whether the product passes (label '0') or fails (label '1') the quality check, having as input tabular data of the assembly process. In this dataset, the features are anonymized, and the only information available about them is to which station and to which line they belong.

Case study 2: CiP learning factory The second case study uses manufacturing data recorded at the Center for Industrial Productivity (CiP), a learning factory from the Institute for Production Management, Technology and Machine Tools of the Technical University of Darmstadt. The resulting dataset, the CiP Discrete Manufacturing Dataset (CiP-DMD), encompasses a comprehensive description of the individual manufacturing processes and the quality measurements corresponding to the production of pneumatic cylinders and its components (cylinder bottom and piston rod). The graph-based approach will be evaluated by developing a predictive quality solution to indicate if the produced cylinder bottoms pass or fail the quality control.

The data analyses performed in this work were computed using an AMD Ryzen 9 3900X 3.8GHz 12-Core CPU with 32GB RAM and an Nvidia RTX A6000 GPU.

5. Graph representation learning for predictive quality in discrete manufacturing

The successful deployment of ML models for a particular task relies heavily on the quality of the data representation, also known as features, employed in their application. In this context, a robust data representation incorporates relevant priors or assumptions, guiding the model towards more effective learning [9].

This thesis introduces a novel data modeling approach to incorporate priors into the ML model, particularly addressing feature dependencies. This approach utilizes graph representation learning to construct rich data representations within a manufacturing context. By leveraging a graph representation, relationships among workstations and their features are explicitly defined, offering a comprehensive representation of interdependencies within the data. Consequently, this representation allows the GNN to account for these relationships during the classification process, a capability not encoded or fed to alternative ML algorithms, as outlined in Section 2.4.2. These graph representations facilitate the classification of products into two categories: Those adhering to defined quality standards (labeled as '0' for OK products) and those deviating from the standards (labeled as '1' for NOK products). Classification is preferred in this evaluation over anomaly detection since the goal is to discern patterns within the data indicative of defective parts. The objective extends beyond recognizing characteristics of good parts and identifying which parts do not present them; it encompasses identifying faults that frequently arise in the production process and mapping the process data with the product's quality characteristics.

As delineated in preceding sections, this research focuses specifically on multi-stage, discrete manufacturing processes. Moreover, its scope is delimited to examining tabular and time series data, assuming that predictive insights into product quality can be derived from data collected throughout the manufacturing process.

5.1. Proposed approach

Figure 5.1 summarizes and visualizes the proposed approach, delineated into six stages. The primary objective is to facilitate the development of a graph representation learning method using manufacturing data, subsequently comparing it with the baselines delineated in Section 3.

This approach draws inspiration from the framework outlined in [15], particularly the concept of a *Product 360*, which represents a comprehensive, 360-degree view of a product. In this framework, all available and pertinent information about a manufactured product is consolidated within a heterogeneous graph.

Analyzing a multi-stage manufacturing process of $n > 1$ workstations or processes, where raw materials are transformed into a final product, the initial step '*I Identify*' involves identifying the Minimal Processing Steps (MPS) within the production system. An MPS is defined here as the smallest step of a process for which data is systematically collected. For instance, within a machining process, an MPS can be defined as a sub-operation such as cutting specific parts of the geometry.

Once the MPSs are established, the subsequent task entails identifying the process variables—aspects of the process from which data is collected. Finally, the relationships between different MPS are identified. This involves comprehending how the sources generating data are interconnected and determining how these

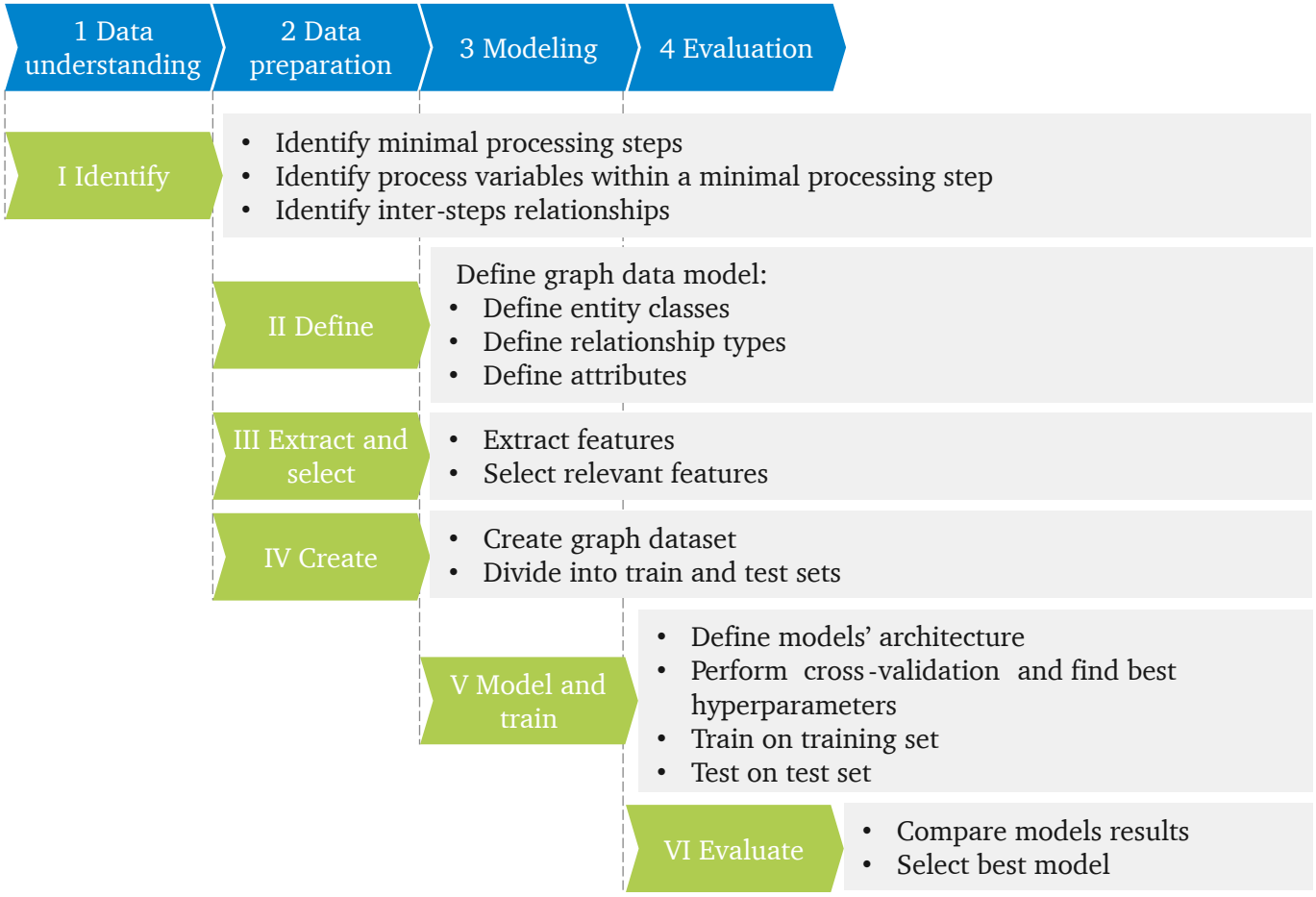


Figure 5.1.: Proposed approach

connections can be effectively represented within the graph, providing a comprehensive understanding of the data structure and relationships.

The data model is formalized in the second step '*II Define*', encompassing decisions regarding defining entity classes, relationship types, and their respective attributes. In this phase, objects and their connections within the domain will be represented as a data model, defining the graph structure that will receive data in the subsequent steps. Every entity or object within the domain is systematically assigned to a distinct entity class, ensuring shared characteristics regarding relationships and attributes. Likewise, relationship types outline their properties and specify the corresponding entity classes to which they refer.

In the case studies explored in this work, data is recorded in either time series or tabular formats. In the third step '*III Extract and select*', features are extracted and selected for each MPS.

The feature extraction encompasses the data transformations as presented in Section 2.3.1. This step precedes feature selection and is particularly relevant for time series data, where capturing meaningful patterns requires additional processing. Both feature extraction and selection are performed for each MPS, as the introduction of variations and process disturbances during the multi-stage process is unpredictable. This approach ensures a holistic understanding of data dynamics at each manufacturing process step.

The feature extraction and selection for time series data are performed using the Python library Time Series FeatuRe Extraction based on Scalable Hypothesis tests (tsfresh) [25]. The tsfresh library automatically extracts diverse features from time series data. These features capture statistical characteristics, trends, patterns, and

other relevant information embedded in the time series. The library employs hypothesis tests to identify which features are statistically significant and likely to be informative. Hypothesis testing helps to select a subset of relevant features, reducing the dimensionality of the dataset while retaining valuable information, as explained in Section 2.3.1.

Once numerical features are available for each produced product and a data model is established to define the connections and characteristics of entities in the graph, the subsequent phase '*IV Create*' involves creating the graph dataset and partitioning it into training and testing sets. The graph dataset comprises numerous graphs, each accompanied by its corresponding labels. To accomplish this, the Python Deep Graph Library (DGL) [110] is employed for graph creation and dataset preparation. This library is also used in the subsequent phase '*V Model and train*', where the GNN architecture is delineated. DGL is purpose-built for constructing and handling graph-structured data in deep learning applications. It provides an interface for the creation and transformation of graphs, and training of GNNs, aligning with the requirements of the task at hand. It is essential to notice that the GNN and baselines models are trained and tested with the same features and numerical dataset. The only difference for the GNN is representing the numerical data in heterogeneous graphs.

The search for optimal model parameters for both the GNN and baseline models employs cross-validation, adhering to the evaluation framework illustrated in Figure 5.2 [96]. After identifying the best model parameters through cross-validation, the model undergoes training on the entire training set and is subsequently tested on the designated test set. The model's performance is then assessed and compared to the results achieved by the baseline models in the last stage '*VI Evaluate*'.

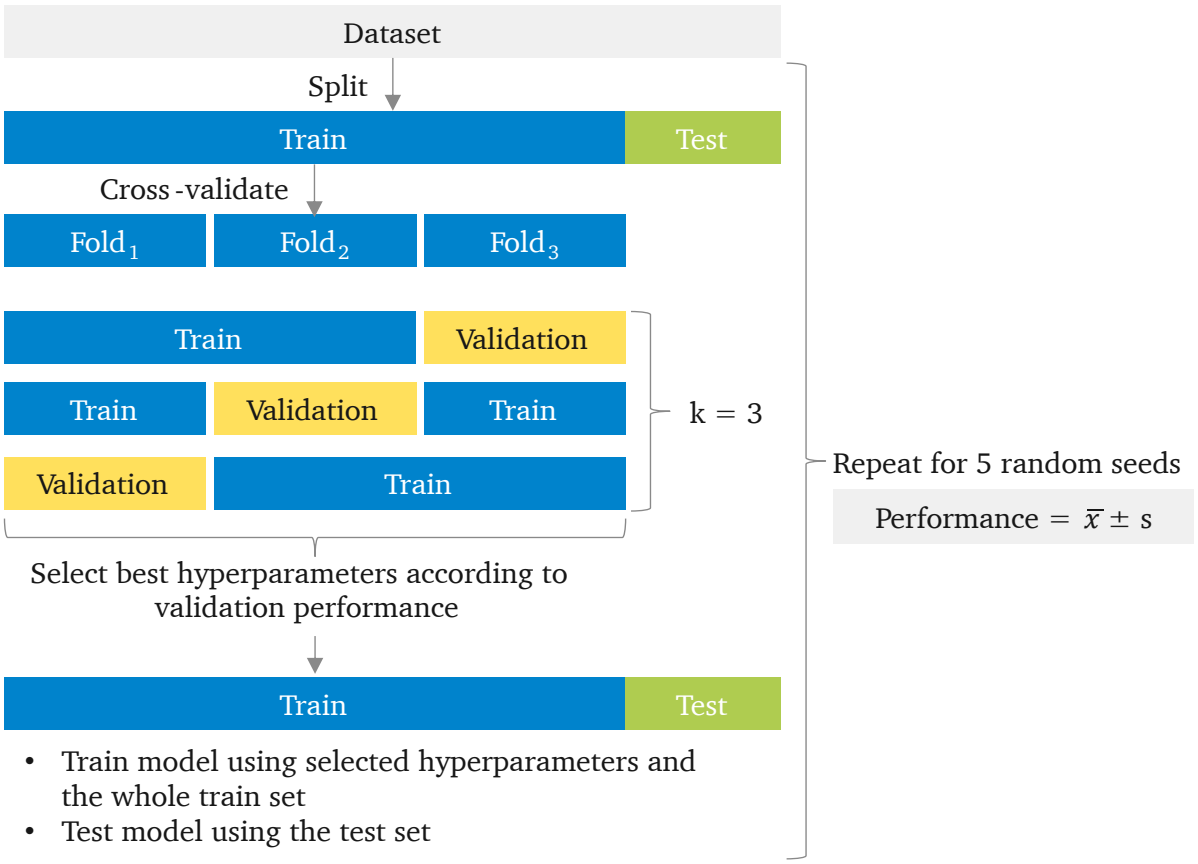


Figure 5.2.: Evaluation framework based on [96]

This approach provides practical guidance and organizes the presentation of the case studies, with each phase delineated for every individual case study. Importantly, it can be applied to other datasets that adhere to the specified assumptions, extending its applicability beyond the specific cases investigated in this study.

The general GNN's model architecture implemented in the case studies is illustrated in Figure 5.3. The GNN model has also been implemented using the Python library DGL. The input to the model are the graphs (or a batch of graphs) composed of the graph structure and the nodes' feature vectors represented by 'h'. The model is composed of a linear layer (a) responsible for changing the node features from their input size to the defined hidden feature size (which is a hyperparameter). The graph convolution layers represented by (b) are a series of sequential graph convolutions applied to the graph followed by an activation function. The number of convolutional layers is also a hyperparameter. The second linear layer (c) transforms the node features from the hidden size to the output size of each node feature vector. The readout layer (d) performs the readout of feature nodes (which can be a sum, mean or maximum operation) and generates the graph-level representation. Finally, the last linear layer (e) gives the output of the GNN model, serving as a mapping from the learned graph representations to a single scalar value, which can be interpreted as the log odds or logits. The model's prediction is obtained by applying the sigmoid function (f) to the output of this linear layer. The sigmoid function squashes the logits into the range $[0, 1]$, representing the probability of belonging to the positive class (class 1). Mathematically, this is expressed as:

$$P(y = 1) = \frac{1}{1 + e^{-\text{logits}}} \tag{5.1}$$

After applying the sigmoid function, the resulting probability is the model's confidence in assigning the positive class (class 1) to a given example. If the probability is close to 1, the model is confident that the example belongs to class 1, while a probability close to 0 indicates confidence in class 0.

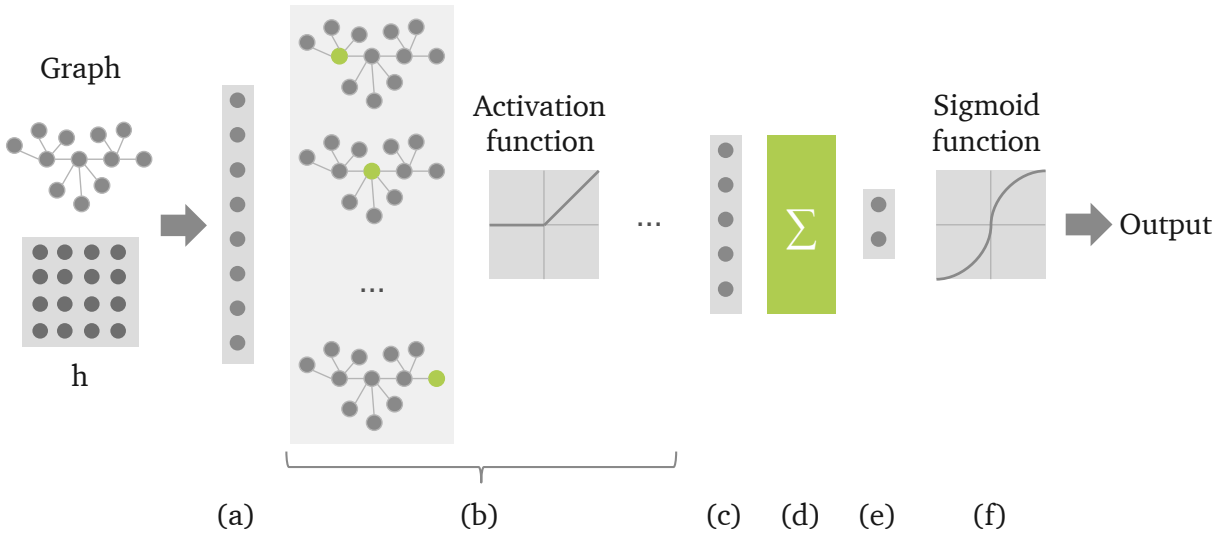


Figure 5.3.: GNN model architecture: (a) linear layer, (b) graph convolution layers, (c) linear layer, (d) readout layer, (e) linear classifier, (f) sigmoid function

The baseline ML algorithms were selected based on the results from Chapter 3, namely LR, RF, SVM, kNN, XGBoost, and MLP. They have been implemented using the Python library Scikit-learn [66]. Below, a short explanation of the hyperparameters of the GNN and baseline models is provided. The hyperparameter values were chosen based on those used in the investigated literature and through experimentation, as presented in Chapter 6. This process ensures that the models are configured with parameters that have demonstrated effectiveness in similar tasks and are further fine-tuned based on the specifics of the case studies.

Table 5.1.: Hyperparameters for GNN and baseline models

Model	Hyperparameters	Explanation
GNN	batch_size	Size of each mini-batch during training
	learning_rate	Learning rate for optimization
	weight_decay	L2 regularization term to prevent overfitting
	scheduler_gamma	Learning rate decay factor
	num_layers	Number of graph convolutional layers
	hid_feats	Number of hidden features in each layer
	aggregator	Aggregation method for neighborhood information
	feat_drop	Dropout rate for node features during training
	activation	Activation function for hidden layers
	conv_type	Type of graph convolutional layer
LR	C	Regularization parameter. Controls the inverse of the regularization strength. Smaller values specify stronger regularization
	penalty	Regularization term. 'l1' or 'l2' specifies the type of penalty applied
	solver	Algorithm to use for optimization
RF	n_estimators	Number of trees in the forest
	max_depth	Maximum depth of the tree. Controls overfitting
	min_samples_split	Minimum number of samples required to split an internal node
	min_samples_leaf	Minimum number of samples required to be at a leaf node
SVM	C	Regularization parameter. Trades off correct classification of training points against maximization of the decision function's margin
	kernel	Specifies the kernel type. 'linear', 'poly', 'rbf', or 'sigmoid' are options
	degree	Degree of the polynomial kernel function (if the kernel is 'poly')
	gamma	Kernel coefficient for 'rbf', 'poly', and 'sigmoid'
kNN	n_neighbors	Number of neighbors to use for classification
	weights	Weight function used in prediction. 'uniform' assigns equal weight, 'distance' weights points by the inverse of their distance
	p	Power parameter for the Minkowski metric. 1 is Manhattan distance, 2 is Euclidean distance
	algorithm	Algorithm used to compute nearest neighbors. 'auto,' 'ball_tree', 'kd_tree', or 'brute' are options
XGBoost	n_estimators	Number of boosting rounds
	learning_rate	Step size shrinkage used in update to prevent overfitting
	max_depth	Maximum depth of a tree
MLP	hidden_layer_sizes	Tuple representing the number of neurons in each hidden layer
	activation	Activation function for the hidden layer. 'relu' or 'tanh' are possible choices
	alpha	L2 penalty (regularization term) parameter

6. Validation

This chapter validates the proposed approach by applying it to two manufacturing datasets through comprehensive case studies. The primary objective of this validation chapter is to assess how well the proposed approach performs compared to the baseline algorithms and to demonstrate its applicability and versatility across different contexts.

Section 6.1 introduces the case study conducted using the Bosch production line performance dataset, offering insights into its application and outcomes. Section 6.2 presents the case study conducted at the CiP learning factory, exploring its unique challenges and findings. Section 6.3 identifies the similarities and differences between the two case studies, providing a comprehensive comparative analysis. Lastly, Section 6.4 draws practical recommendations, facilitating applying the proposed approach in diverse contexts. Overall, this chapter provides empirical evidence and validates the proposed approach through rigorous examination and evaluation.

6.1. Case study 1: Bosch production line performance

The first case study assesses the proposed methodology using the open-source dataset *'Bosch production line performance.'* This dataset was published on the website Kaggle in 2016 and is presented in the form of tabular data stored in Comma Separated Values (CSV) files [13]. The dataset contains measurements of parts as they are sequentially assembled in Bosch's assembly lines. The measurements are organized in columns of anonymized features indicating the line and station where each measurement was recorded. Each part is uniquely identified by a specific number found in the column 'Id'. The complete dataset is subdivided into categorical, date, and numeric datasets.

Bosch released this dataset as part of a competition aimed at identifying the most effective ML model for predicting parts that would fail quality control, denoted by a 'Response = 1' in the dataset. The features within the dataset adhere to a naming code that indicates the line, the station, and the feature number. For instance, *L3_S36_F3939* designates a feature measured on line 3, station 36, with feature number 3939. It is important to note that, due to this anonymization, the nature of the process characteristics being measured by these features remains undisclosed.

This dataset was chosen as the first case study for three main reasons. Firstly, it encompasses data derived from a multi-stage discrete manufacturing process, fitting the scope of this thesis. Second, although the features are anonymized, the tabular format of the data and its naming convention permits the identification of stations and features, along with their interdependencies, for each assembled product. Additionally, the open-source nature of this dataset allows benchmarking the proposed approach and comparing it with subsequent approaches or improvements. The goals of the experiments with the Bosch dataset can be summarized as follows:

1. Verify the impact different graph data models have on the performance of a single product variant.

2. Verify whether the graph representation of the data and its classification with a GNN achieve better results than the specified baselines for a single product variant.

An initial proof of concept was developed with the Bosch numeric dataset, which allowed validating the approach before exploring further datasets. Part of this section's content has been published as a paper titled "Multi-source data modelling and graph neural networks for predictive quality" [16]. Section 6.1.1 provides further details about the numeric dataset and how it has been prepared for evaluating the graph representation learning approach following the guidelines provided in Figure 5.1.

6.1.1. Data understanding and preparation

The complete train numeric dataset is tabular data containing 1,183,747 rows and 970 columns. Table 6.1 displays the 'Id' and 'Response' columns and the first six feature columns for the first ten rows of the dataset. The 'Id' column contains a unique identifier for each assembled product. The last column of the dataset called 'Response' contains the label indicating whether the product passed (label equals '0') or failed (label equals '1') the end-of-line quality control. From the total amount of assembled products, 6,879 are labeled as '1', which results in 0.58% of NOK products. As previously mentioned, the remaining column names are anonymized and follow the pattern LX_SX_FX, where X indicates the number of the line (L), station (S), or feature (F). This pattern allows us to identify the number of lines, the stations within each line, and the number and assignment of features to each station.

Table 6.1.: First ten rows of numeric dataset

Id	L0_S0_F0	L0_S0_F2	L0_S0_F4	L0_S0_F6	L0_S0_F8	L0_S0_F10	…	Response
4	3	-34	-197	-179	118	116	…	0
6							…	0
7	88	86	3	-52	161	25	…	0
9	-36	-64	294	33	74	161	…	0
11	-55	-86	294	33	118	25	…	0
13	3	19	294	312	31	161	…	0
14							…	0
16							…	0
18	-16	-41	-179	-179	-56	161	…	0
23							…	0

The complete dataset has three assembly lines, 50 stations, and 968 features. Figure 6.1 illustrates the stations, including their number and sequence in each assembly line. Station 37 from line 3, has the most extensive number of NOK products (6,556 NOK products have passed through this station), followed by station 29 (6,295 NOK products), and station 34 (5,718 NOK products) both from line 3.

Initially, the dataset is grouped into product variants, defined as products that have traversed identical lines and stations with the same feature types. A product variant is identified in the dataset through the presence or absence of a numeric value for a specific column and row. Figure 6.2 shows the ten variants with the largest number of products and the number of defective products in each of them.

Variant 3, identified as having the highest number of defective products among variants with significant sample sizes, was selected for experimentation. The decision to focus on the predictive quality approach for this variant was guided by practical considerations and a strategic choice to address a variant with a substantial impact on the manufacturing process. This choice was motivated by the need to develop insights and solutions that could directly benefit the production line's efficiency.

The selected variant comprises 11,592 samples, with 86 (0.74%) classified as non-conforming (NOK) products. This variant represents approximately 9.8% of the entire dataset, making it a strategically important

Figure 6.1.: Lines and stations in the numeric dataset

subset for investigation. Focusing on a variant with a notable proportion of defects provides an opportunity to develop a predictive quality model to identify and mitigate issues specific to this product variant effectively. In addition to the strategic importance of the chosen variant, the decision to target a specific variant also reduces computational complexity. Analyzing the entire dataset, with multiple variants and a diverse range of features, could introduce unnecessary computational overhead. The research thus streamlines the analysis by narrowing the focus to variant 3.

Table 6.2 provides a detailed breakdown of the distribution of lines and stations, along with the number of features per station. This table offers a comprehensive overview of the assembly process being analyzed. As shown in the figure, variant 3 traverses three lines, encompassing eight stations and a total of 210 features.

Table 6.2.: Number of features for each line and station

Line	Station	Number of features
L1	S24	49
L2	S26	14
L3	S29	53
L3	S30	68
L3	S33	10
L3	S34	4
L3	S35	8
L3	S37	4

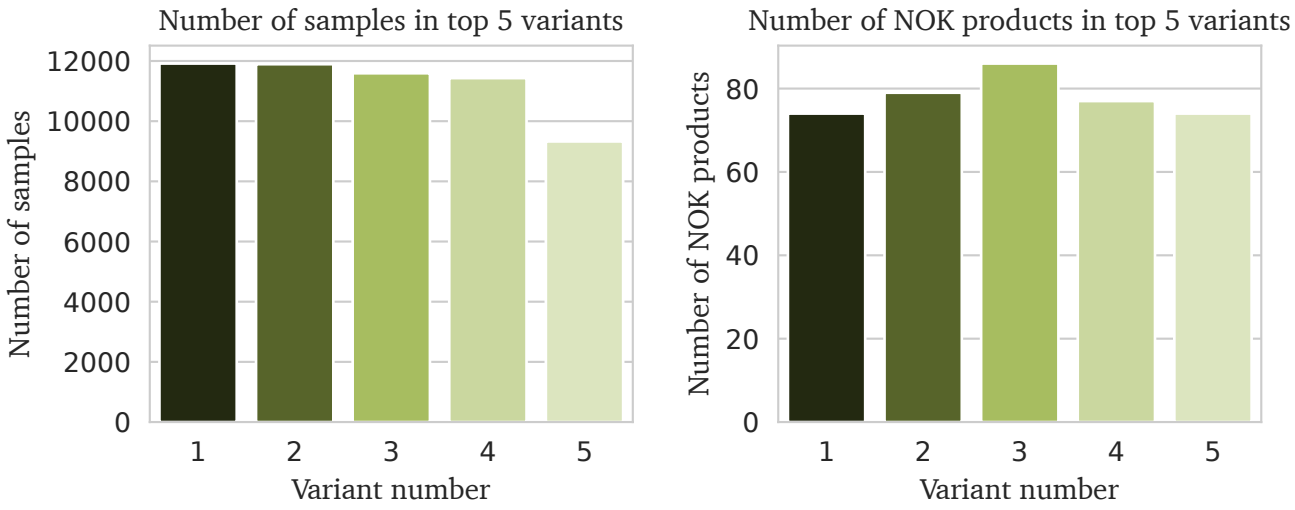


Figure 6.2.: Number of samples (left) and number of NOK parts (right) for the top five variants with the largest number of products in the train numeric dataset

I Identify The Bosch numeric dataset portrays an assembly workflow of distinct stations, identified as the MPS for this dataset. Within each station, individual features represent process variables associated with that station. This arrangement forms a hierarchical structure where lines occupy the highest level, succeeded by stations and features. Stations are interconnected with their subsequent stations, reflecting the movement of the semi-assembled products between these stations.

II Define Eight graph data models are proposed to represent the identified processing steps, variables, and relationships. Figure 6.3 visually represent the eight data models and their respective variations. The data models differ in three key aspects:

1. Entity classes:
   - Models 1 and 2 have three entity classes: Line, Station, and Feature.
   - In contrast, models 3 and 4 have two entity classes: Station and Feature.
2. Relationship types and direction:
   - Models labeled with types 1 and 3 have edges flowing from the higher concept in the hierarchy to the lower concept (e.g., from Station to Feature).
   - Conversely, models labeled with types 2 and 4 have edges flowing from the lower concept to the higher concept in the hierarchy (e.g., from Feature to Station).
3. Feature attributes:
   - Type ‘a’ models represent the entity class ‘Feature’ attributes as vectors containing both the feature value and the one-hot-encoded identification of that feature.

- On the other hand, type 'b' models represent the entity class 'Feature' attributes as vectors containing only the feature value.

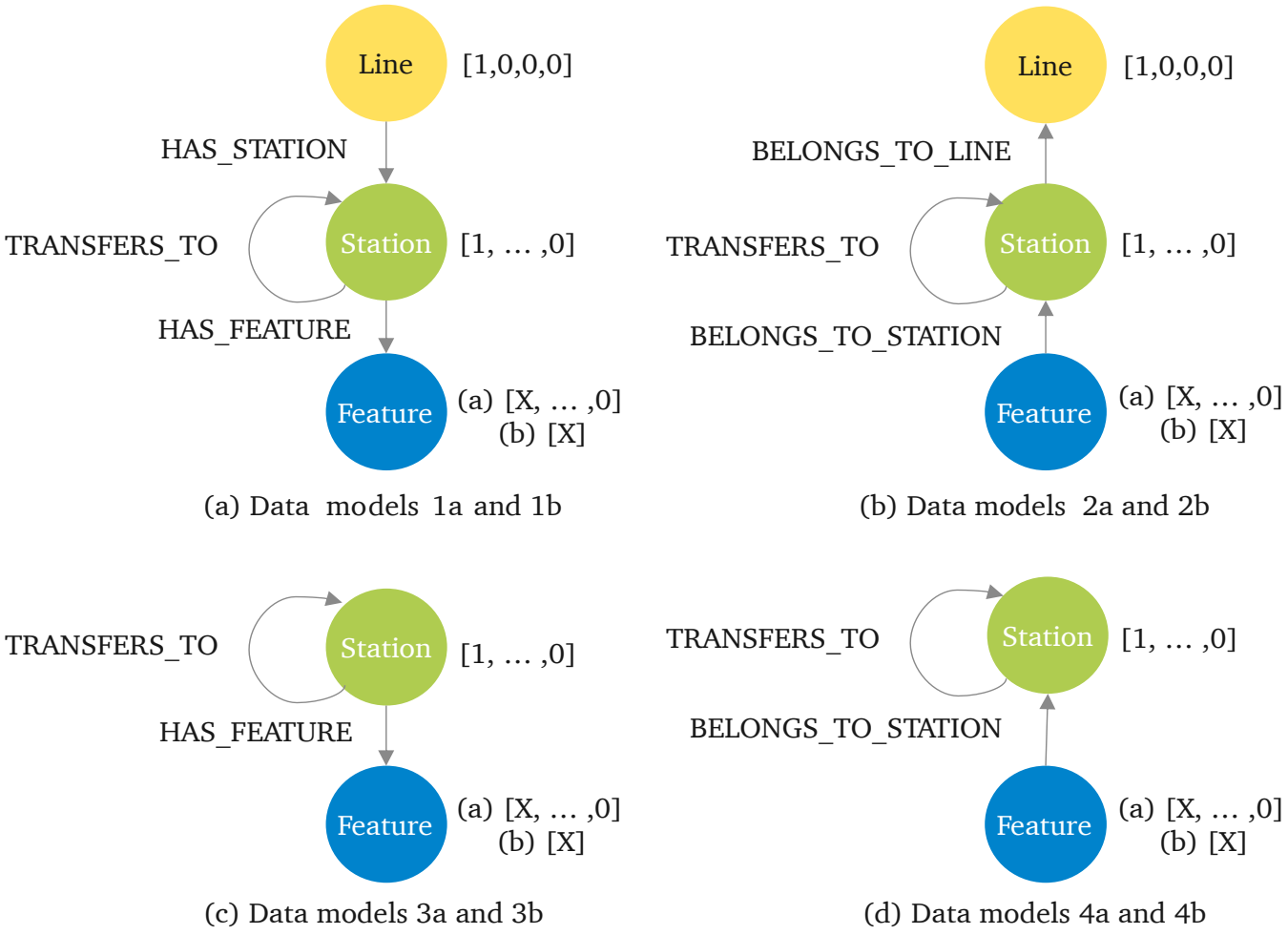


Figure 6.3.: Data models 1a, 1b, 2a, 2b, 3a, 3b, 4a and 4b

These data models are first contrasted based on their performance in an initial step of the evaluation, presented in Section 6.1.2.

III Extract and select The features are selected for dimensionality reduction. Initially, features exhibiting a substantial presence of zero values, surpassing 80%, are removed from the dataset. This initial filtration step aims to exclude features that predominantly contribute negligible information.

Subsequently, the remaining features are evaluated, and three features per station (the largest possible number of features to ensure that all stations have the same number of features) are selected using univariate tests, with their p-values calculated employing the tsfresh Python library. This step ensures that the chosen features possess significant statistical relevance to construct a refined dataset for robust model training. It is important to emphasize that the baseline and GNN models are trained on identical tabular datasets comprising the same numerical features. However, the tabular dataset is transformed into a graph-based representation before being fed into the GNN models. This transformation aims to encapsulate the relationships within the

data, enabling the GNN models to consider the interdependencies and patterns among lines, stations, and features.

IV Create Graph datasets are generated for each of the eight data models based on the selected features according to the procedure described in Algorithm 7.

Table 6.3 presents a snippet of the first five rows and four columns of variant 3's dataset. The column names and values serve as entities for identifying lines, stations, and feature classes, with the 'featureValue' attributing to the feature nodes. This translation from the tabular representation to a graph dataset involves capturing essential identifiers from the column names and values. These identifiers help in defining nodes within the graph. Specifically, they represent distinct entities such as lines, stations, and feature classes.

Table 6.3.: Excerpt of variant 3's tabular dataset

Id	L1_S24_F1604	L1_S24_F1632	L1_S24_F1846
127	-8	-17	-145
137	-8	2	-94
389	-8	-55	-196
400	-8	-3	-126
496	-8	-61	-205

Table 6.4 shows the data of the first product in the dataset, identified as product ID 127. This table facilitates the identification of the entities along with their corresponding names, showcasing the relationships 'HAS_STATION' and 'HAS_FEATURE', as well as 'BELONGS_TO_LINE' and 'BELONGS_TO_STATION'. The presented information provides an overview of the interconnections within the dataset.

Table 6.4.: Entities and feature values for product ID 127

lineId	stationId	featureId	featureValue
L1	L1_S24	L1_S24_F1604	-8
L1	L1_S24	L1_S24_F1632	-17
L1	L1_S24	L1_S24_F1846	-145
L2	L2_S26	L2_S26_F3073	-125
L2	L2_S26	L2_S26_F3106	58

The relationship 'TRANSFERS_TO' was obtained from the station sequence in the dataset. Table 6.5 shows the resulting table with the station of origin (stationId_s, 's' for the subject) and the destination (stationId_o, 'o' for the object).

Table 6.5.: TRANSFERS_TO relationship

stationId_s	stationId_o
L1_S24	L2_S26
L2_S26	L3_S29
L3_S29	L3_S30

Algorithm 7: Data preprocessing and graph creation—Bosch case study

```
if data_model is 1a or 1b or 2a or 2b then
    Define Entity classes:
        Line;
        Station;
        Feature;
else
    Define Entity classes:
        Station;
        Feature;
Procedure preprocess_data():
    Read raw data;
    Identify Line, Station or Feature nodes according to data model;
    Identify edges between Line, Station, and Feature nodes according to data model;
Procedure load_data_to_graph(data_model):
    Create empty graph g;
    Edges creation if data_model is 1a or 1b then
        Create edges for Line to Station, Station to Station, and Station to Feature relationships;
    else if data_model is 2a or 2b then
        Create edges for Station to Line, Station to Station, and Feature to Station relationships;
    else if data_model is 3a or 3b then
        Create edges for Station to Feature and Station to Station relationships;
    else if data_model is 4a or 4b then
        Create edges for Feature to Station and Station to Station relationships;
    Node features creation if data_model is 1a or 2a then
        Add node features for Line, Station and Feature nodes as one-hot-encoded features to the graph;
    else if data_model is 1b or 2b then
        Add node features for Line, Station and Feature nodes to the graph;
    else if data_model is 3a or 4a then
        Add node features for Station and Feature nodes as one-hot-encoded features to the graph;
    else if data_model is 3b or 4b then
        Add node features for Station and Feature nodes to the graph;
Procedure main():
    Read configuration parameters (data model, data source, etc.);
    Preprocess data using preprocess_data;
    Create graphs and load data to the graph using load_data_to_graph;
    Save created graphs in graph dataset;
```

These auxiliary tables were used to create the graph and load the numerical features into it. Figure 6.4 shows the resulting graph for product ID 127. In this example, the data model 1a was used. The relationship types 'HAS_STATION', 'HAS_FEATURE' and 'TRANSFERS_TO' can be recognized by the direction of arrows in the graph. The complete graph dataset contains 11,592 graphs and their respective labels obtained from the 'Response' column of the numeric dataset. Section 6.1.2 explains the modeling and training processes for the GNN models as well as the baselines and associated hyperparameters.

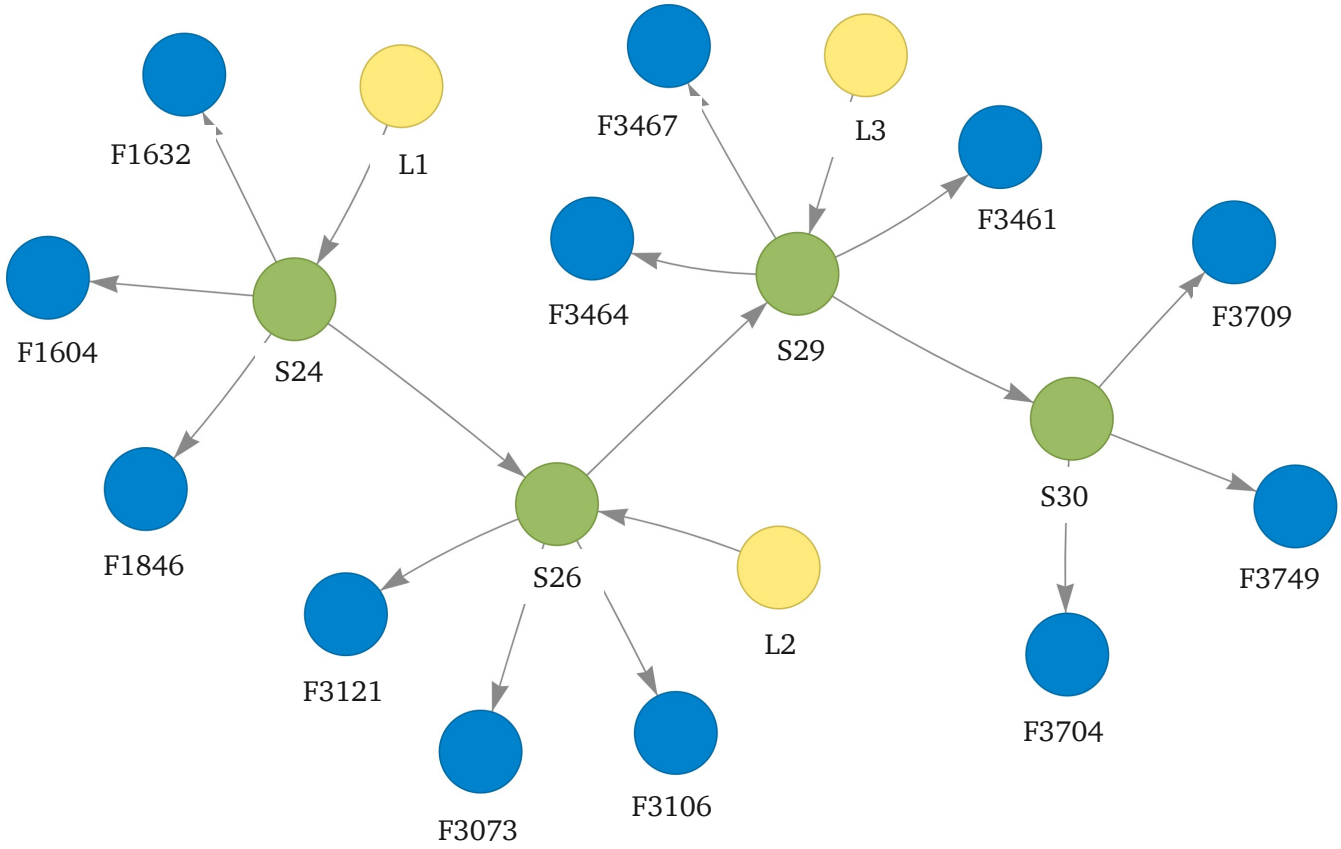


Figure 6.4.: Resulting graph for product ID 127 (data model 1a)

6.1.2. Modeling and evaluation

V Model and train The GNN's model architecture follows the same general structure as shown in Figure 5.3. This architecture is adapted to process graphs of each data model. Below are the hyperparameters employed in both the GNN and the baseline models, they define the specific settings of the models employed in this case study.

Table 6.6.: Hyperparameters for GNN and baseline models

Model	Hyperparameters	Hyperparameters values
GNN	batch_size	[128, 256]

Table 6.6 (continued)

Model	Hyperparameters	Hyperparameters values
	learning_rate	[0.001, 0.0001]
	weight_decay	[0.0001, 0.00001]
	scheduler_gamma	[1, 0.995, 0.9]
	num_layers	[1, 3, 6]
	hid_feats	[64, 128]
	aggregator	[mean, pool, lstm]
	feat_drop	[0.1, 0.3, 0.4]
	activation	[F.relu, F.leaky_relu, F.elu]
	conv_type	[sage, gin, conv]
LR	C	[0.1, 1, 10]
	penalty	[l1, l2]
	solver	[liblinear, saga]
RF	n_estimators	[50, 100, 200]
	max_depth	[5, 10, 20]
	min_samples_split	[2, 4, 5]
	min_samples_leaf	[1, 2]
SVM	C	[0.1, 1.0, 100]
	kernel	[linear', poly', rbf', sigmoid']
	degree	[2, 3, 4]
	gamma	[scale, auto]
kNN	n_neighbors	[3, 5, 7]
	weights	[uniform', distance']
	p	[1, 2]
	algorithm	[auto, ball_tree, kd_tree, brute]
XGBoost	n_estimators	[50, 100, 150]
	learning_rate	[0.05, 0.1, 0.2]
	max_depth	[2, 3, 4]
MLP	hidden_layer_sizes	[(50,), (100,), (200,)]
	activation	[relu, tanh]
	alpha	[0.0001, 0.001, 0.01]

The model takes as input the data model, along with specifications such as the size of hidden and output features, the activation function, the number of layers, and the type of graph convolution layer. The available graph layers include GraphConv, GraphSAGE, and GIN, as outlined earlier. The models' readout function is the summation of all node features.

The cross-validation, training, and testing procedures adhere to the evaluation framework presented in

Figure 5.2. The test size is 20% of the complete dataset, while the remaining 80% constitutes the training dataset, which is also used for cross-validation. Data division between training and test datasets is performed for each random seed. To assess the impact of different data models on model performance, the initial evaluation is performed for a single random seed. The data model exhibiting the best performance is then selected for comparison with the baseline models. The number of epochs for the GNN models varies between 5 and 10. This range prevents overfitting and ensures that the model's performance with the training set remains comparable to the test set's performance.

VI Evaluate Eight different GNN models are trained with the datasets respective to each data model. The results are displayed in Table 6.7. Data model 4b stands out as the best-performing model across multiple metrics. It has the highest values for Precision, F_1 score, AUC-ROC, and MCC. The high F_1 score and MCC suggest a good balance between precision and recall. Therefore, data model 4b is selected for being contrasted with the baseline algorithms.

Table 6.7.: Data models performance comparison. *Best performing model

Data model	Precision	Recall	F_1 score	AUC-ROC	MCC
1a	1.6%	41.2%	3.1%	61.3%	5.0%
1b	1.6%	52.9%	3.1%	64.3%	5.7%
2a	3.4%	35.3%	6.3%	64.0%	9.0%
2b	9.5%	25.0%	13.8%	61.7%	14.5%
3a	0.7%	100.0%	1.5%	50.0%	0.0%
3b	18.0%	50.0%	3.5%	65.8%	6.7%
4a	8.9%	29.4%	13.7%	63.6%	15.1%
4b*	**71.4%**	**31.2%**	**43.5%**	**65.6%**	**47.0%**

The GNN and baseline models are trained and tested for five random seeds following the cross-validation framework illustrated in Figure 5.2. Table 6.8 provides the performance of each algorithm across three metrics: F_1 score, AUC-ROC, and MCC. The asterisk (*) indicates the best-performing model for each metric. Notably, the GNN outperforms other models for two out of three metrics. The column with the standard deviations provide insights into each metric's variability or uncertainty in model performance.

Table 6.8.: Performance comparison—Product variant number 3. *Best performing model

Algorithm	F_1 score	Standard deviation
LR	2.9%	0.8%
RF	26.8%	12.6%
SVM	10.5%	7.6%
kNN	15.8%	2.5%
XGBoost	24.3%	10.9%
MLP	14.8%	6.1%
GNN*	**36.0%**	**4.6%**
Algorithm	**AUC-ROC**	**Standard deviation**
LR*	**66.8%**	**2.6%**
RF	60.5%	6.0%

Table 6.8 (continued)

Algorithm	F_1 score	Standard deviation
SVM	59.7%	5.9%
kNN	61.0%	3.6%
XGBoost	61.5%	6.1%
MLP	60.6%	5.9%
GNN	65.8%	2.4%
Algorithm	**MCC**	**Standard deviation**
LR	13.2%	15.9%
RF	28.0%	12.3%
SVM	11.6%	7.9%
kNN	16.0%	3.1%
XGBoost	24.5%	11.4%
MLP	15.0%	6.8%
GNN*	**37.2%**	**7.6%**

- F_1 score: The GNN is the top performer, with an F_1 score of 36.0% alongside a standard deviation of 4.6%. Following behind is the RF model, which achieves a performance level of 26.8% with a standard deviation of 12.6%. Notably, the GNN outperforms its nearest competitor, the RF model, by approximately 34% in F_1 score. The GNN model exhibits a reduction of around 36% in standard deviation compared to the RF model, indicating a higher level of stability.

- AUC-ROC: The LR model demonstrates the highest performance in terms of AUC-ROC, achieving 66.8%, with a standard deviation of 2.6%. Following LR, the GNN model is the second-best performer, with an AUC-ROC of 65.8% and a lower standard deviation of 2.4%. Despite its slightly lower AUC-ROC than LR, the GNN model exhibits more stability, which is evident from the lower standard deviation. These results suggest that while LR may have a marginally higher mean performance, the GNN model offers greater consistency in its predictions.

- MCC: The GNN model is the top performer, with an MCC of 37.2%, alongside a notably lower standard deviation of 7.6%. The RF and XGBoost models are the second and third best-performing models, with MCC scores of 28.0% and 24.5%, respectively, and standard deviations ranging from 11.4% to 12.3%. Regarding MCC, the GNN model outperforms the second-best performing model, the RF model, by approximately 33%. The standard deviation of the GNN model is approximately 38% lower than that of the RF model.

Figure 6.5 illustrates the precision-recall curves, providing a visual representation of the performance of each evaluated algorithm. As explained in Section 2.3.3, these curves offer insights into the trade-off between precision and recall at various classification thresholds, offering a comprehensive view of algorithmic effectiveness for a specific random seed. This specific plot reveals that all models face challenges achieving both high precision and high recall simultaneously. The GNN curve is positioned closer to the top right corner, indicating a more favorable balance between precision and recall compared to other algorithms. This suggests that, in the context of the evaluated task, the GNN exhibits a better overall performance in capturing relevant instances while minimizing false positives.

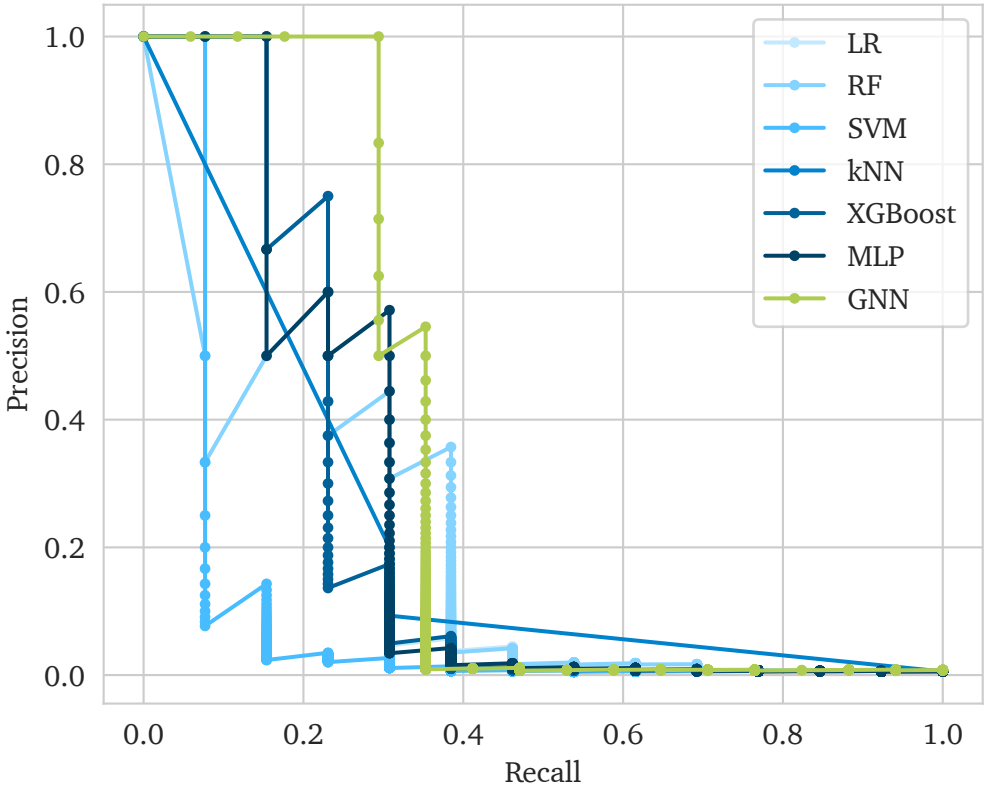


Figure 6.5.: Precision-recall curves

6.1.3. Discussion

The relatively low performance of all models can be attributed to characteristics of the Bosch production line performance dataset, including the complexity of the prediction task and the quality of the data itself, such as the presence of noise and irrelevant features, which can adversely impact the models' ability to learn meaningful patterns. The problem is inherently challenging and lacks clear patterns, so all models struggle to achieve high performance. Nevertheless, these experiments help evaluate the models' discriminative power and generalization capability, providing valuable insights into their strengths and limitations. Notably, the GNN model outperforms other models, presenting the highest F_1 and MCC scores.

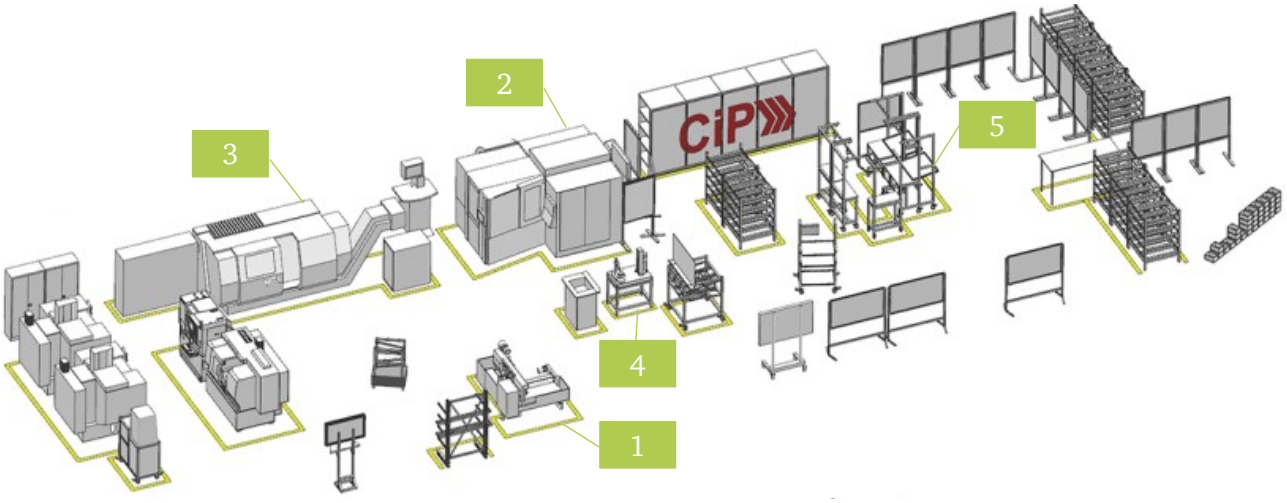
Objective 1: impact of different graph data models To understand the impact of different graph data models on the performance of a single product variant, eight different GNN models were trained with datasets corresponding to each data model. The results in Table 6.7 highlight data model 4b as the best-performing model across multiple metrics. This model exhibited the highest values for precision, F_1 score, AUC-ROC, and MCC, suggesting a well-balanced performance. Consequently, Data Model 4b was selected for further comparison with baseline algorithms.

Objective 2: GNN's performance compared to baselines The GNN emerges as the top performer for two out of three metrics, exhibiting a marginal difference of approximately 1.5% from the leading model in the AUC-ROC metric and the lowest standard deviation across all metrics, indicating superior prediction stability. These findings highlight the GNN model's enhanced capacity for discerning underlying patterns within the dataset compared to the baseline models. Additionally, the precision-recall curves in Figure 6.5 offer insights into the trade-off between precision and recall, emphasizing the GNN's more favorable balance in capturing relevant instances.

6.2. Case study 2: CiP learning factory

The second case study explores the CiP Discrete Manufacturing Dataset (CiP-DMD) [57], a dataset comprising recordings of a multi-stage discrete manufacturing process for producing a pneumatic cylinder. The dataset was recorded at the Center for Industrial Productivity (CiP) and includes data from the manufacturing processes of two components of the pneumatic cylinder (the cylinder bottom and the piston rod) and the final product assembly with additional supplied parts. The author of this thesis is one of the contributors to the data collection and writing of the paper presenting the CiP-DMD [57].

The CiP is a learning factory of the PTW at the Lichtwiese campus of the TUDa in Germany. A learning factory embodies a conceptual framework encompassing processes in a specified setting resembling a value chain, a manufactured physical product, and a didactic concept [1]. It serves as a dynamic learning environment where individuals being trained can gain insights from their actions within this simulated factory setting. This holistic approach integrates hands-on experiences with theoretical learning, providing a comprehensive educational platform within manufacturing and production processes. The production hall of the CiP learning factory, illustrated in Figure 6.6, is equipped with (1) a sawing machine Kasto sba A2, (2) a milling machine Deckel Maho DMC-50H, (3) a turning machine Index C65, (4) a measuring station, and (5) an assembly line.



1 = Sawing machine Kasto SBA2, 2 = Milling machine DMC 50H,
3 = Turning machine Index C65, 4 = Measuring station, 5 = Assembly line

Figure 6.6.: Layout of the CiP learning factory at PTW, TU Darmstadt

Figure 6.7 shows the data acquired within the CiP that is relevant for this thesis, where process data is marked in green and quality data is highlighted in blue. A traceability system was implemented within the CiP to link the two data types. This system involved assigning unique identification numbers to each manufactured part, imprinting them on adhesive tags, and utilizing scanners at designated checkpoints in the manufacturing process to capture timestamps and link the relevant information. During the production weeks necessary for recording the dataset, students assumed the roles of manufacturing workers. Their responsibilities encompassed a wide range of tasks, including operating machinery, scanning identification numbers on individual parts, conducting measurements at the measuring station, and assembling the final product. The subset of the CiP-DMD used for validating the proposed approach contains data of the manufacturing process of 824 cylinder bottoms and their quality measurements.

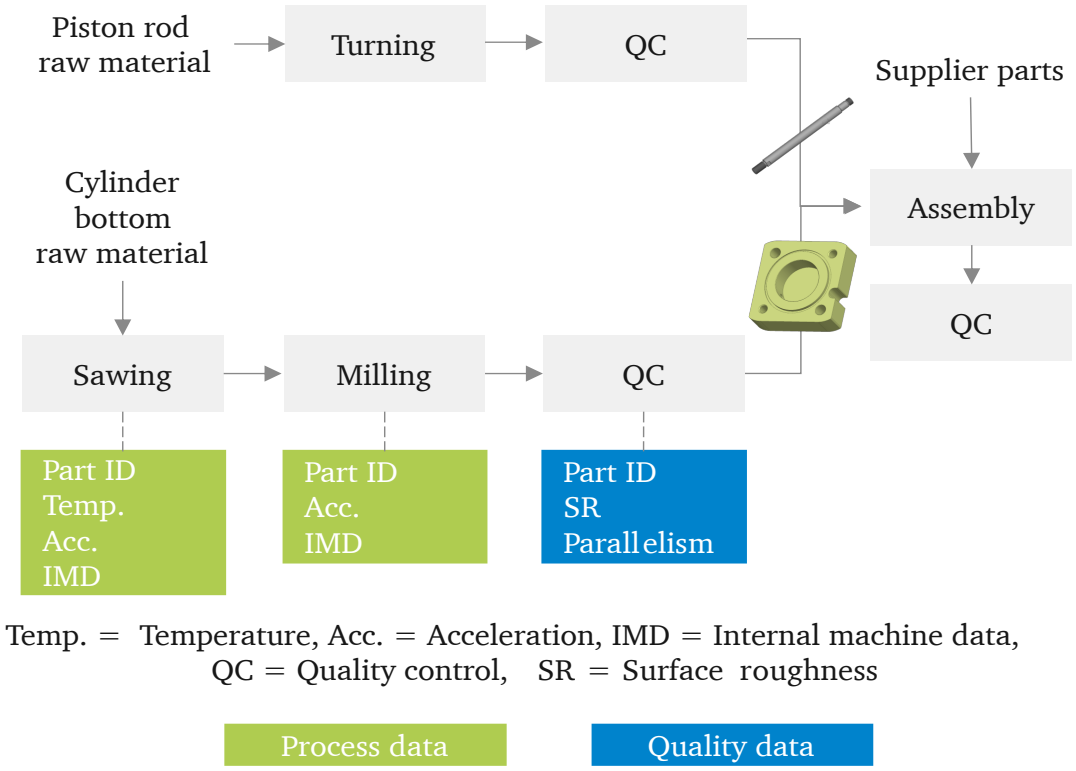


Figure 6.7.: CiP-DMD data acquisition relevant to this thesis

Process data The manufacturing process of the cylinder bottom can be subdivided into 14 sub-processes: (1) sawing, (2) face cutting, (3) outer contour roughing and finishing, (4) groove side milling, (5) step boring, (6) end burring and external contour boring, (7) lateral drilling, (8) drilling sinking, (9) drilling, (10) thread milling, (11) face cutting, (12) circular pocket milling, (13) deburring, and (14) ring groove milling. For a detailed description of the manufacturing process of the cylinder bottom please refer to Appendix A. The first sub-process is conducted on the sawing machine, and the remaining sub-processes are done on the milling machine. First, the raw material consisting of square steel bars is cut in the sawing machine, as shown in Figure 6.8.

The sawing machine is equipped with temperature sensors, a position sensor for recording the saw's arm position, vibration sensors, a photoelectric sensor for measuring the belt speed, a pressure sensor, and a current transformer, which are recorded with a sampling rate of 5 Hz. Besides external sensors, internal machine data, such as the cutting counter and cutting time counter, are also recorded with the same sampling rate of 5 Hz. The data is continuously sent to a database via Open Platform Communications Unified Architecture (OPC UA). The processing time for cutting the square steel bars into one blank is 143 seconds.

The blanks are then transported to the milling machine. The machining area contains a rotating tower with fixtures to hold the blanks and allow the machining of each side of the cylinder bottom. At each cycle, a blank and a semifinished part are clamped on the tower at opposite sides. At the end of a cycle, which lasts five minutes (199 seconds for the front side and 113 seconds for the back side), the milling machine outputs one finished and one semifinished cylinder bottom, as presented in Figure 6.9. The milling machine has been retrofitted with a triaxial accelerometer mounted on the spindle and connected to an industrial computer. The accelerometer data is recorded with a sampling rate of 2.5 kHz. Internal machine data, such as the Numerical Control (NC) line and spindle current, is accessed and recorded with a sampling rate of 5 Hz using the Siemens Brownfield Connectivity gateway via Message Queuing Telemetry Transport (MQTT) protocol.

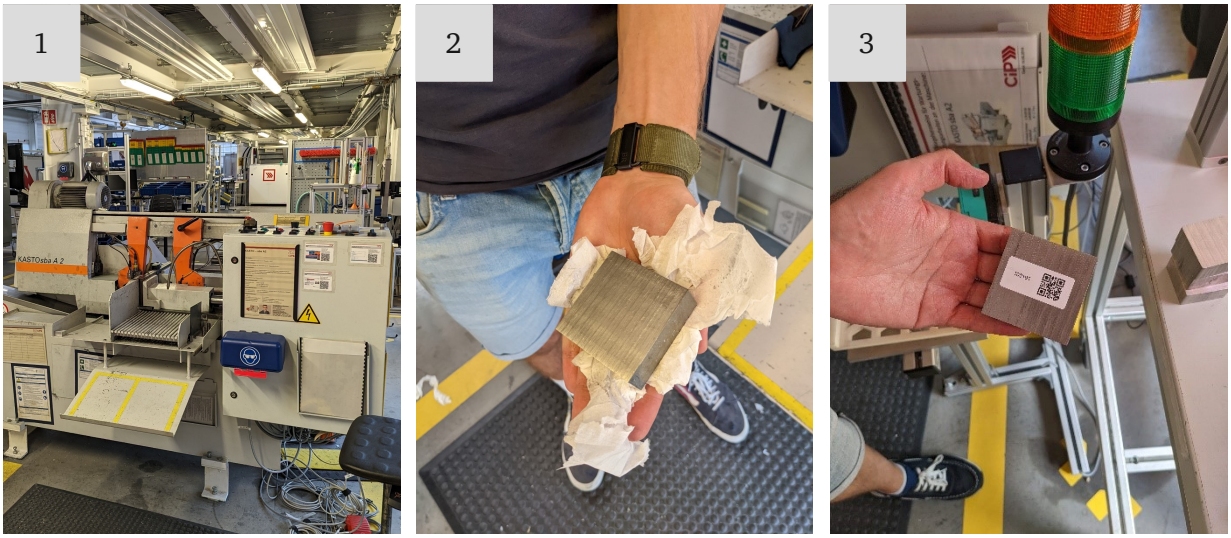


Figure 6.8.: (1) Sawing machine Kasto sba A2. (2) Steel blank after sawing. (3) Adhesive tag with identification number and QR code attached to each blank

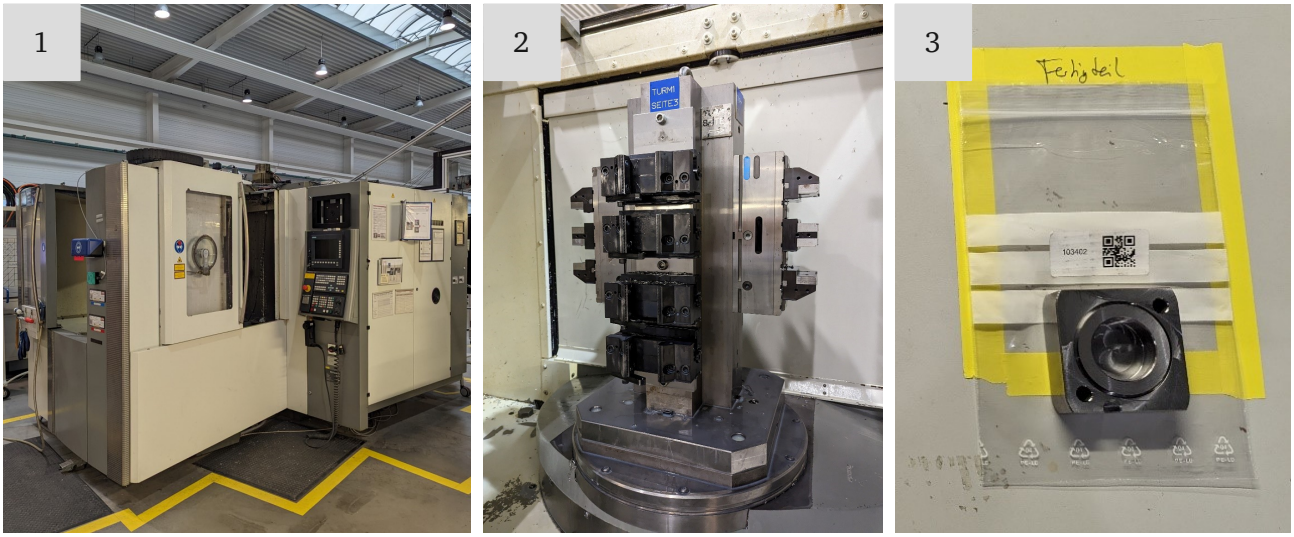


Figure 6.9.: (1) Milling machine Deckel Maho DMC-50H. (2) Rotating tower with fixtures to hold the blanks and semifinished cylinder bottom. (3) Cylinder bottom after machining processes

Quality data The quality control of the cylinder bottom is performed at the measuring station after milling operations are completed. The measuring station is equipped with a Mahr MarSurf M400 for measuring the surface roughness and a Millimess 2001 for measuring the parallelism (see Figure 6.10). The quality control is performed manually by the students and the values are automatically recorded using the Q-DAS procella software. The quality measurements from the quality control indicated in Figure 6.7 are measurements of the cylinder bottoms' surface roughness and parallelism. Table 6.9 contains the dimensional specifications for each quality measurement.

Table 6.9.: Dimensional Specifications

Dimension	Target value	Lower limit	Upper limit
Surface roughness	0 µm	0 µm	2.5 µm
Parallelism	0 mm	0 mm	0.05 mm

Three controlled anomalies were introduced in the cylinder bottom's manufacturing process. The first anomaly was introduced in the sawing process: 99 blanks were cut too short. Cutting off a blank too short resulted in parts with a lower weight and caused further anomalies in the subsequent machining process in the milling machine. The second anomaly, a consequence of the first, is related to machining the too-short blanks. Processing these blanks causes the face milling step not to remove any chips, resulting in lower measured accelerations. The third anomaly type is the false clamping of 40 blanks in the milling tower. This anomaly causes an uneven removal of chips on the back side, producing crooked cylinder bottoms. The second anomaly type directly affects the surface roughness (measured in the front of the cylinder bottom), and the third affects the parallelism. The labels regarding the surface roughness measurements for the binary classification were defined based on the allowed tolerance range of 2.5 µm, with a target value of 0 µm. Values within the tolerance range were labeled as '0', and values outside as '1'. Out of the 824 bottoms assessed, 637 were labeled '0', indicating conformity, while 187 were labeled '1', denoting non-conformity. This corresponds to a percentage of approximately 22.7% NOK (not meeting quality standards) parts. The labels regarding the parallelism measurements for the binary classification were defined based on the allowed tolerance range of 0.05 mm, with a target value of 0 mm. Values within the tolerance range were labeled as '0', and values outside as '1'. Out of the 824 bottoms assessed, 755 were labeled '0', indicating conformity, while 69 were labeled '1', denoting non-conformity. This corresponds to a percentage of approximately 8.4% NOK parts. Table 6.10 presents the breakdown of OK and NOK parts among the 824 cylinder bottoms in the dataset for each quality measurement.

Table 6.10.: Quality control labels

Quality measurement	OK	NOK
Surface roughness	637 (77.3%)	187 (22.7%)
Parallelism	755 (91.6%)	69 (8.4%)

For three reasons the CiP-DMD dataset was selected as the second case study. Firstly, it fulfills the requirements of encompassing data from a multi-stage discrete manufacturing process, aligning with the focus of this thesis. Secondly, the dataset is non-anonymized, providing transparency into all processes and quality evaluations, which can be effectively modeled as a graph. Moreover, being open-source, this dataset facilitates benchmarking of the proposed approach and enables comparisons with subsequent methodologies or

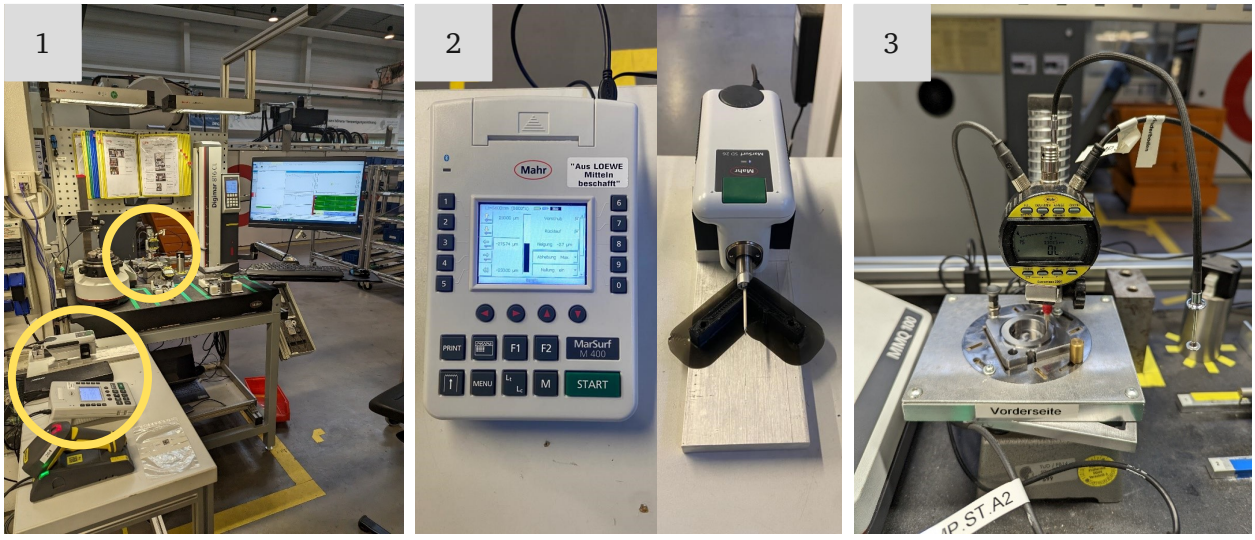


Figure 6.10.: (1) Measuring station with highlighted location of measuring devices. (2) Roughness meter Mahr MarSurf M400. (3) Precision pointer Millimess 2001

enhancements. The objectives of the experiments conducted using the CiP-DMD dataset can be summarized as follows:

1. Assess the impact of different graph data models on the performance of the classification model for the surface roughness and parallelism labels.
2. Assess the impact of the number of features for each sub-process on the classification performance for the surface roughness and parallelism labels.
3. Evaluate whether the graph representation of the data and its classification using a GNN yield superior results compared to specified baselines for the surface roughness and parallelism labels.

These objectives address critical aspects of the research. Firstly, they examine the effectiveness of different graph data models and the number of features in classification performance, providing insights into model optimization. Secondly, they evaluate the performance of the proposed approach using GNN against established baselines, contributing to the validation and potential adoption of the methodology. These objectives directly support the thesis's goal of developing and validating a robust classification model for manufacturing process quality evaluation.

6.2.1. Data understanding and preparation

I Identify The MPS for the CiP case are the sub-processes for which data has been acquired, specifically the 14 sub-processes (sawing and 13 sub-processes from the milling process) outlined previously. Within each sub-process, the process variables comprise the data recorded from individual sensors and machine internal data, detailed in Table 6.11. The relationships between steps are defined by the sequential progression from sawing to milling and the connections between process variables associated with each respective sub-process.

Table 6.11.: Overview data acquisition and sub-processes CiP-DMD

Process	Processing time	Acquired data	Sampling rate
Sawing	143 seconds	Part ID	1 value/part
		Temperature	5 Hz
		Acceleration	5 Hz
		Arm position	5 Hz
		Band speed	5 Hz
		Pressure	5 Hz
		Current	5 Hz
		Internal machine data	5 Hz
Milling	312 seconds	Part ID	1 value/part
		Acceleration	2.5 kHz
		Internal machine data	5 Hz

II Define Four graph data models have been proposed to represent the identified processing steps, variables, and their relationships. These models contain two primary entity classes: Process and Feature. The Process class represents the sub-processes conducted at the sawing and milling machines, while the Feature class is dedicated to representing extracted features derived from the process variables. These features may include statistical measures like mean and maximum temperature values observed during a sub-process. In Figure 6.11, the four distinct data models are illustrated alongside their respective variations for the CiP case study. The differences among these models are evident in two aspects:

1. Relationship types and direction:
    - Models labeled with the type ‘1’ have edges flowing from the higher concept in the hierarchy to the lower concept (from Process to Feature).
    - Conversely, models labeled with the type ‘2’ have edges flowing from the lower concept to the higher concept in the hierarchy (from Feature to Process).
2. Feature attributes:
    - Type ‘a’ models represent the entity class ‘Feature’ attributes as vectors containing both the feature value and the one-hot-encoded identification of that feature.
    - On the other hand, type ‘b’ models depict the entity class Feature attributes as vectors containing only the feature value.

These differences are illustrated in Figure 6.12 showing part of the manufacturing process of the cylinder bottom and three examples of feature values both for the data models ‘1a’ and ‘1b’.

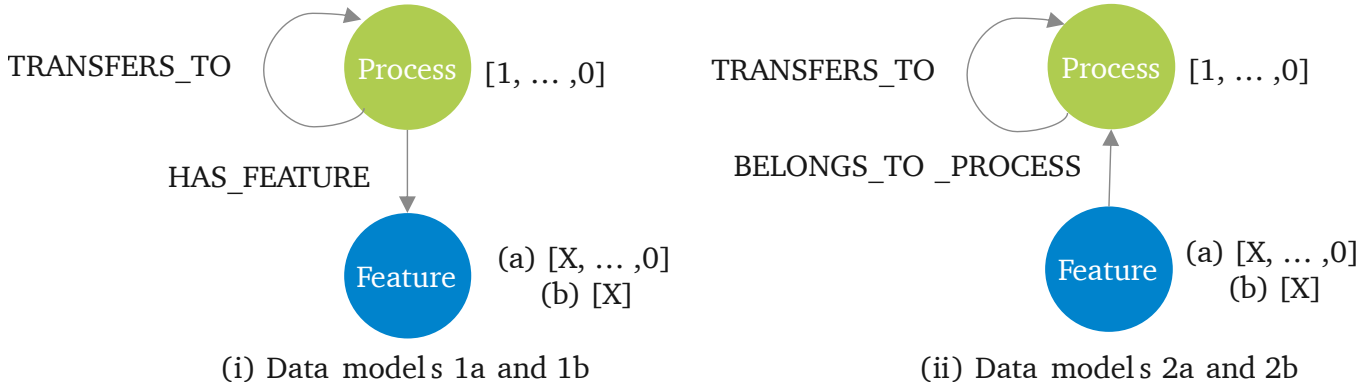


Figure 6.11.: Data models 1a, 1b, 2a and 2b

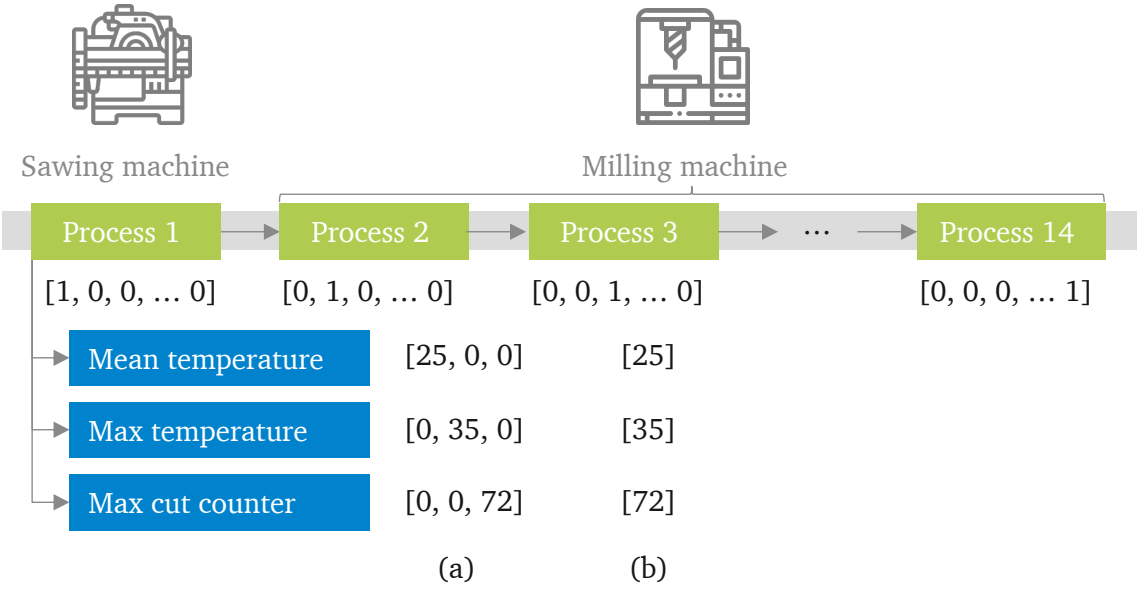


Figure 6.12.: Illustration of data models 1a and 1b for CiP case study

III Extract and select Statistical and frequency features are extracted from the time series data of each sub-process utilizing the Python library tsfresh [25]. Initially, sets of three and five features are chosen for each sub-process to evaluate the impact of feature quantity. If a performance enhancement is observed moving from three to five features, ten features are then selected for each sub-process. Increasing the number of features from three to five and then to ten is a strategic approach to optimize model performance while balancing computational efficiency. If transitioning from three to five features leads to noticeable performance enhancements, it suggests that additional features capture relevant information about the underlying processes more effectively. Thus, expanding the feature set to ten features further refines the data representation, potentially capturing nuanced patterns and improving predictive accuracy. By incrementally increasing the number of features, the experimentation process becomes iterative, allowing for an exploration of the feature space's potential. Detailed information regarding which features were extracted and selected for each sub-process and quality label is provided in Appendix B.

The experiments are repeated for both the GNN and the baseline models. The most pertinent features for each sub-process are determined via the same feature selection method from the tsfresh library used in the first case study, which evaluates the significance of the extracted features. Specifically, each feature's influence

on the target (the column containing the labels) is assessed through univariate tests, with p-values calculated accordingly. This feature extraction and selection process is repeated for each quality measurement, namely surface roughness and parallelism.

IV Create Graph datasets are constructed for the four data models and the chosen features following the pseudocode outlined in Algorithm 8. It is emphasized again that identical data is utilized for both training and testing the GNN and baseline approaches. The only distinction lies in the presentation of the data, as it is inserted into a graph specifically for training the GNN approach. This methodology ensures robustness in assessing the efficacy of the GNN and underscores the reliability of the comparative analysis.

Algorithm 8: Data preprocessing and graph creation

```
Define Entity classes:
    Process;
    Feature;
Procedure preprocess_data():
    Read raw data;
    Identify Processes and Features nodes;
    Identify edges between Features and Processes nodes;
Procedure load_data_to_graph(data_model):
    Create empty graph g;
    Edges creation: if data_model is 1a or 1b then
        Create edges for Process to Feature and Process to Process relationships;
    else if data_model is 2a or 2b then
        Create edges for Feature to Process and Process to Process relationships;
    Node features creation: if data_model is 1a or 2a then
        Add node features for Process and Feature nodes as one-hot-encoded features to the graph;
    else if data_model is 1b or 2b then
        Add node features for Process and Feature nodes to the graph;
Procedure main():
    Read configuration parameters (data model, data source, etc.);
    Preprocess data using preprocess_data;
    Create graphs and load data to the graph using load_data_to_graph;
    Save created graphs in graph dataset;
```

Figure 6.13 shows the resulting graph for the first product of the dataset. In this example, the data model 1a was used. The relationship types 'HAS_FEATURE' and 'TRANSFERS_TO' can be recognized by the direction of arrows in the graph. Figure 6.14 offers a closer view of part of the graph, in which the sub-processes and their features can be recognized. The complete graph dataset contains 824 graphs and their respective labels obtained from the surface roughness and parallelism measurements. Section 6.2.2 explains the modeling and training processes for the GNN models and the baselines and associated hyperparameters.

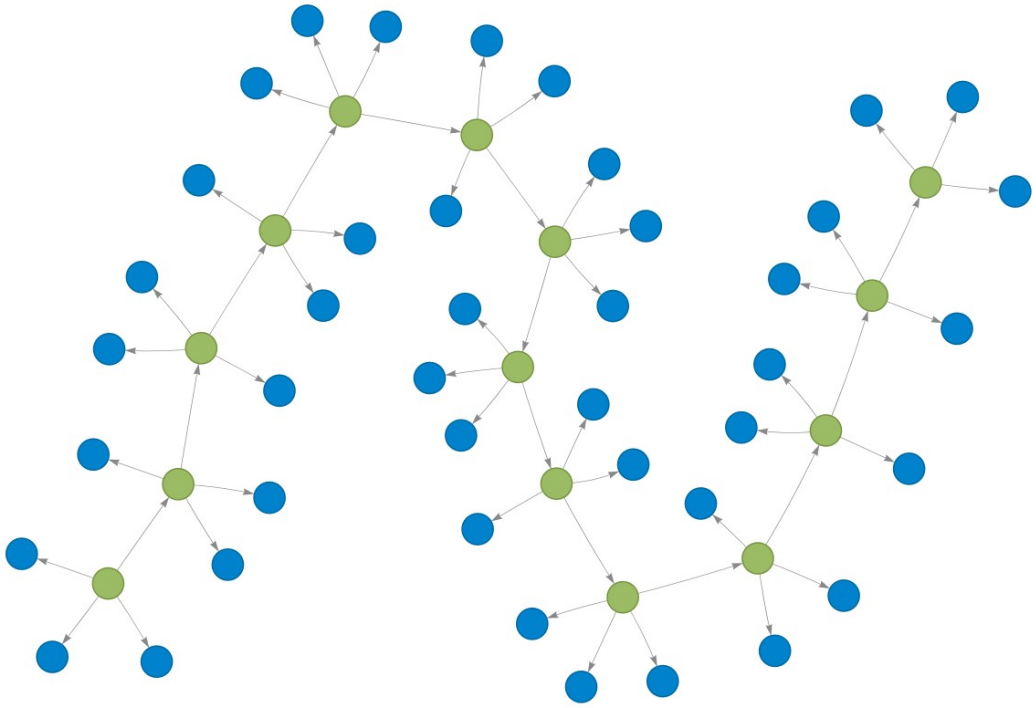


Figure 6.13.: Resulting graph for first product in the dataset (data model 1a)

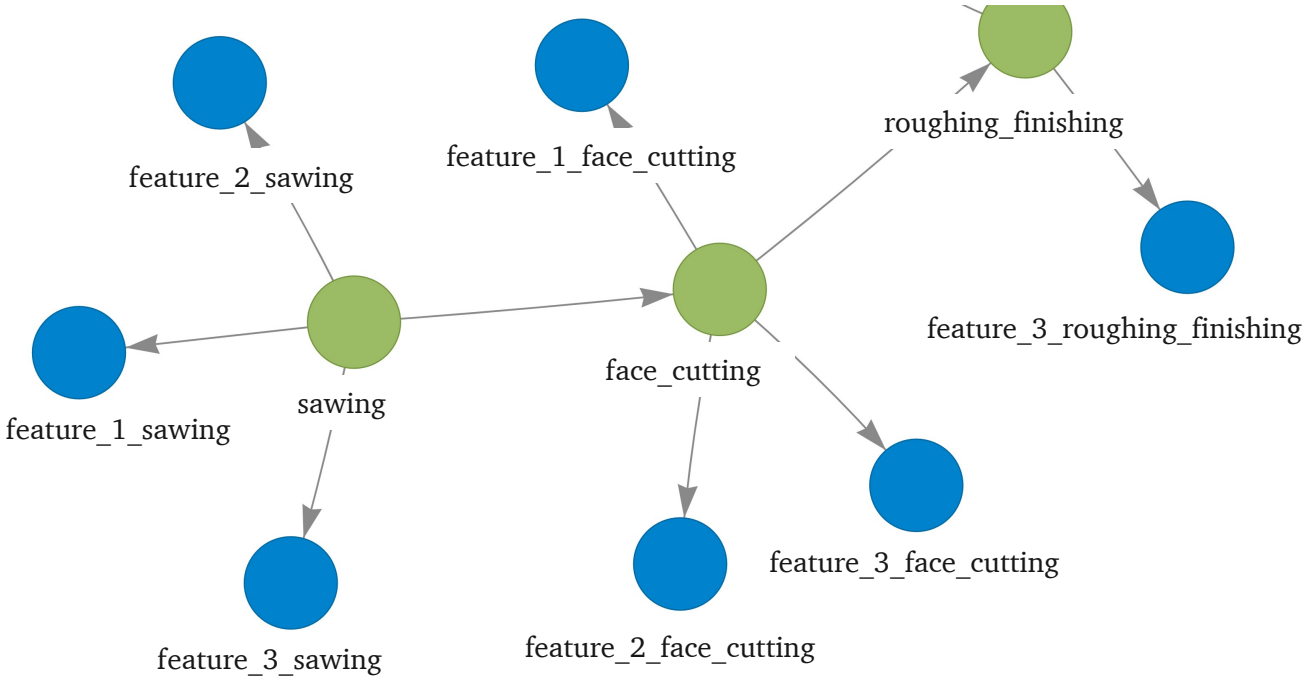


Figure 6.14.: Resulting graph for first product in the dataset (data model 1a)—Detailed view

6.2.2. Modeling and evaluation

V Model and train The architecture of the GNN model consists of a series of linear and graph convolutional layers. These layers systematically operate on node feature vectors, employing a sequential message-passing framework explained in Section 2.4.2. Through this process, information is propagated to generate graph-level representations of the data, effectively capturing the graph's topological structure and attribute-based information. The final layer of the GNN is a linear layer responsible for determining the probability of the graph belonging to either binary class. Subsequently, a sigmoid function is utilized to map the raw output into a probability score between 0 and 1, facilitating the classification decision by thresholding the probability to assign the binary labels '0' or '1'.

The GNN's model architecture follows the same general structure as shown in Figure 5.3, also used in the first case study. Table 6.12 presents the hyperparameters employed in both the GNN and the baseline models, defining the specific settings of the models employed in this case study.

Table 6.12.: Hyperparameters for GNN and baseline models

Model	Hyperparameters	Hyperparameters values
GNN	batch_size	[128, 256]
	learning_rate	[0.001, 0.0001]
	weight_decay	[0.0001, 0.00001]
	scheduler_gamma	[1]
	num_layers	[4, 5, 6]
	hid_feats	[48, 96]
	aggregator	[mean, gcn, pool, lstm]
	feat_drop	[0]
	activation	[F.relu]
	conv_type	[sage, gin, conv]
LR	C	[0.1, 1, 10]
	penalty	[l1, l2]
	solver	[liblinear, saga]
RF	n_estimators	[50, 100, 200]
	max_depth	[5, 10, 20]
	min_samples_split	[2, 4, 5]
	min_samples_leaf	[1, 2]
SVM	C	[0.1, 1.0, 100]
	kernel	[linear, poly, rbf, sigmoid]
	degree	[2, 3, 4]
	gamma	[scale, auto]
kNN	n_neighbors	[3, 5, 7]
	weights	[uniform', distance']
	p	[1, 2]
	algorithm	[auto, ball_tree, kd_tree, brute]
XGBoost	n_estimators	[50, 100, 150]

Table 6.12 (continued)

Model	Hyperparameters	Hyperparameters values
	learning_rate	[0.05, 0.1, 0.2]
	max_depth	[2, 3, 4]
MLP	hidden_layer_sizes	[(50,), (100,), (200,)]
	activation	[relu, tanh]
	alpha	[0.0001, 0.001, 0.01]

The model takes as input the data model, the number of features per sub-process, along with specifications such as the size of hidden and output features, the activation function, the number of layers, and the type of graph convolution layer. The available graph layers include GraphConv, GraphSAGE, and GIN, as outlined earlier. The model's readout function is the summation of all node features.

The cross-validation, training, and testing procedures adhere to the evaluation framework presented in Figure 5.2. The test and training sizes are the same as in the first case study and correspond to 20% and 80% of the complete dataset, respectively. Data partitioning between training and test datasets is conducted for each random seed. To assess the impact of different data models on model performance, the initial evaluation is performed for a single random seed for each quality label. The data model exhibiting the best performance is then selected for comparison with the baseline models. The number of epochs for the GNN models varies between 5 and 15.

VI Evaluate This stage presents an in-depth evaluation of the performance of the proposed data and GNN models across both quality labels. Through the examination of the experiments' results, the objective is to achieve the outlined goals of this case study while also providing practical recommendations for the effective application of these findings in additional multi-stage processes.

Surface roughness Different GNN models are trained using datasets corresponding to each data model, employing both three and five features per sub-process for the surface roughness label, all initialized with the same random seed. The results are displayed in Table 6.13. Data model 2a consistently outperforms the other models across all evaluation metrics, showing the highest precision, recall, F_1 score, AUC-ROC, and MCC. The lower performance of models 1a and 1b suggests that the relationships' direction significantly influences the model's efficacy. Similarly, Model 2b underperforms relative to model 2a, particularly regarding precision. This suboptimal performance of model 2b implies that the absence of additional information on the feature type, as identified by the one-hot-encoded features present in model 2a, might constrain its ability to identify positive instances accurately. Data model 2a is then chosen for comparison with the baseline algorithms.

Table 6.13.: Data models performance comparison—Surface roughness (3 and 5 features)

Features	Data model	Precision	Recall	F_1 score	AUC-ROC	MCC
3	1a	63.4%	97.0%	76.7%	90.9%	71.6%
3	1b	63.4%	91.6%	75.0%	88.1%	68.0%
3	**2a***	**88.0%**	**100.0%**	**93.7%**	**97.9%**	**91.9%**

Table 6.13 (continued)

Features	Data model	Precision	Recall	F_1 score	AUC-ROC	MCC
3	2b	72.9%	94.6%	82.3%	92.0%	77.3%
5	1a	85.0%	94.4%	89.5%	94.8%	86.4%
5	1b	76.5%	97.2%	85.7%	94.1%	81.7%
5	**2a***	**86.0%**	**100.0%**	**92.5%**	**97.6%**	**90.5%**
5	2b	78.5%	94.3%	85.7%	93.5%	81.8%

The GNN and baseline models are trained and tested for five random seeds following the cross-validation framework illustrated in Figure 5.2. The performance comparison shown in Table 6.14 highlights the effectiveness of the GNN and baseline models in predicting the surface roughness label across three classification metrics: F_1 score, AUC-ROC, and MCC. The GNN model is the top performer across all metrics, achieving F_1 score, AUC-ROC, and MCC scores of 91.5%, 95.5%, and 89.0%, respectively, with relatively low standard deviations of 1.5%, 1.8%, and 1.9%, respectively. The RF model is closely behind, exhibiting competitive performance (90.8%, 94.0%, and 88.1%) with slightly higher standard deviations (2.2%, 1.4%, and 2.8%).

Table 6.14.: Performance comparison—Surface roughness (3 features). *Best performing model

Algorithm	F_1 score	Standard deviation
LR	83.3%	2.7%
RF	90.8%	2.2%
SVM	85.2%	2.5%
kNN	87.4%	4.7%
XGBoost	86.6%	4.1%
MLP	87.7%	0.8%
GNN*	**91.5%**	**1.5%**
Algorithm	**AUC-ROC**	**Standard deviation**
LR	92.3%	2.1%
RF	94.0%	1.4%
SVM	90.2%	%
SVM	90.2%	1.7%
kNN	93.7%	2.3%
XGBoost	91.4%	2.7%
MLP	93.0%	0.8%
GNN*	**95.5%**	**1.8%**
Algorithm	**MCC**	**Standard deviation**
LR	78.7%	3.6%
RF	88.1%	2.8%
SVM	81.0%	3.3%
kNN	83.8%	5.9%
XGBoost	82.7%	5.4%
MLP	84.0%	1.0%

GNN*	**89.0%**	**1.9%**

The experiments were repeated for the datasets with five features for each sub-process, to assess whether more features lead to a performance improvement. Table 6.15 shows the effectiveness of the GNN and baseline models in predicting surface roughness across the selected classification metrics. The GNN model is again the top performer, achieving F_1 score, AUC-ROC, and MCC scores of 90.3%, 95.3%, and 87.5%, respectively, with relatively low standard deviations of 1.2%, 1.6%, and 1.7%, respectively. The XGBoost and MLP models are closely behind, which also exhibit competitive performance with similarly low standard deviations. Since the performance did not improve when increasing the number of features from three to five, the experiments were not repeated for ten features per sub-process.

Table 6.15.: Performance comparison—Surface roughness (5 features). *Best performing model

Algorithm	**F_1 score**	**Standard deviation**
LR	80.1%	3.6%
RF	87.3%	2.9%
SVM	85.2%	1.8%
kNN	85.3%	3.6%
XGBoost	87.4%	1.2%
MLP	87.1%	2.9%
GNN*	**90.3%**	**1.2%**
Algorithm	**AUC-ROC**	**Standard deviation**
LR	89.2%	1.6%
RF	91.4%	2.0%
SVM	90.1%	1.8%
kNN	92.5%	2.1%
XGBoost	91.7%	1.5%
MLP	92.6%	2.2%
GNN*	**95.3%**	**1.6%**
Algorithm	**MCC**	**Standard deviation**
LR	74.2%	4.5%
RF	83.8%	3.6%
SVM	81.0%	2.1%
kNN	81.0%	4.7%
XGBoost	83.9%	1.4%
MLP	83.5%	3.7%
GNN*	**87.5%**	**1.7%**

The precision-recall curves showcased for each model, based on a single random seed as presented in Figure 6.15, further underscore the superiority of the GNN approach. The GNN curve occupies the upper-right quadrant, showing its capability to cover the optimal area and achieve the most favorable balance between precision and recall.

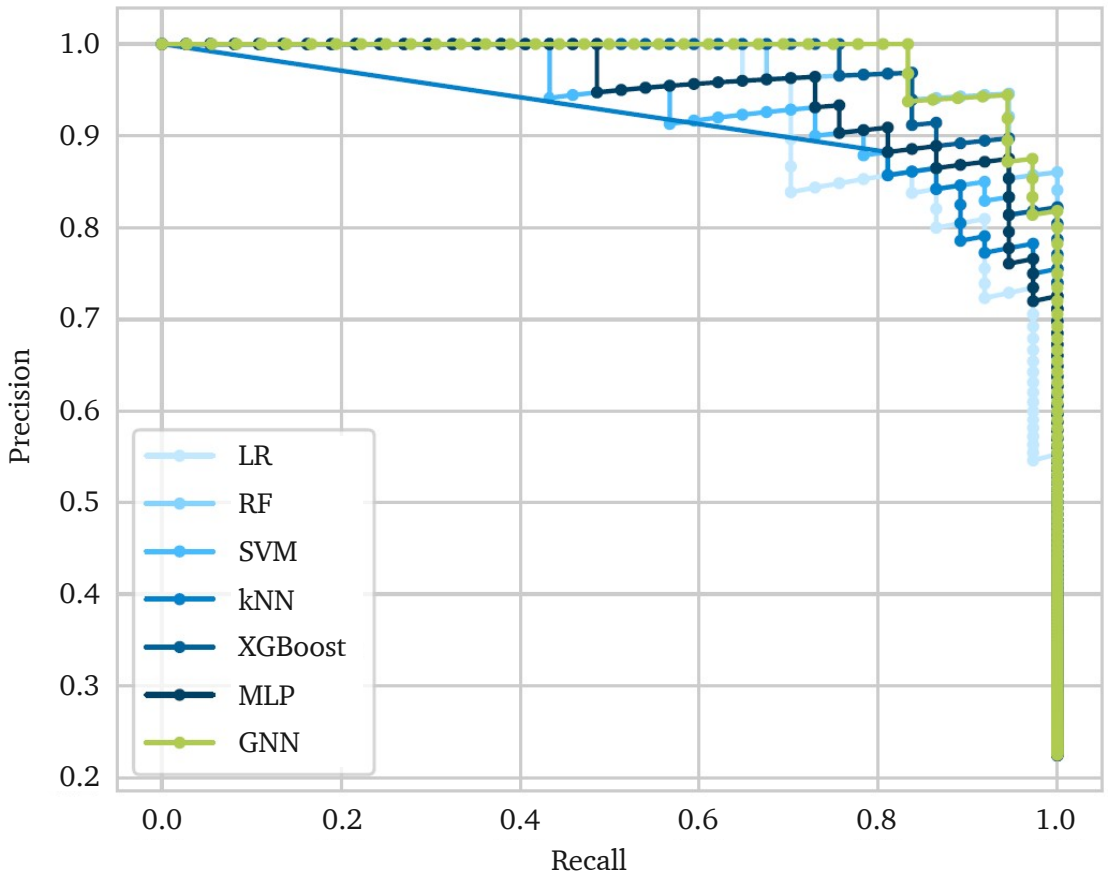


Figure 6.15.: Precision-recall curves—Surface roughness

Parallelism Different GNN models are trained using datasets corresponding to each data model, employing both three and five features per sub-process for the parallelism label, all initialized with the same random seed. The results are displayed in Table 6.16. Data model 2a again outperforms the other data models across all evaluation metrics, showing the highest precision, recall, F_1 score, AUC-ROC, and MCC. The same conclusions can be made regarding the comparison between data models: The lower performance of models 1a and 1b suggests that the relationships' direction significantly influences the model's efficacy. Similarly, the suboptimal performance of model 2b implies that the absence of additional information on the feature type, as identified by the one-hot-encoded features present in model 2a, might constrain its ability to identify positive instances accurately. Data model 2a is again chosen for comparison with the baseline algorithms.

Table 6.16.: Data models performance comparison—Parallelism (3 and 5 features)

Features	Data model	Precision	Recall	F_1 score	AUC-ROC	MCC
3	1a	63.1%	92.3%	75.0%	93.8%	73.9%
3	1b	34.1%	100.0%	50.9%	90.7%	52.7%
3	**2a***	**77.8%**	**100.0%**	**87.5%**	**98.6%**	**87.0%**
3	2b	50.0%	100.0%	66.7%	95.6%	67.5%

Table 6.16 (continued)

Features	Data model	Precision	Recall	F_1 score	AUC-ROC	MCC
5	1a	55.5%	83.3%	66.6%	88.9%	64.9%
5	1b	80.0%	28.6%	42.1%	63.9%	45.3%
5	**2a***	**100.0%**	**76.9%**	**86.9%**	**88.5%**	**86.8%**
5	2b	57.1%	66.7%	61.5%	81.3%	58.4%

The comparative analysis of models for predicting parallelism quality labels with three features per subprocess reveals that the GNN model outperforms other models across multiple evaluation metrics, as shown in Table 6.17. Specifically, the GNN model achieves the highest F_1 score of 79.6% (standard deviation of 4.9%), followed by the XGBoost model with a F_1 score of 77.2% (standard deviation of 10.1%). Similarly, regarding AUC-ROC, the GNN model achieves the highest score of approximately 90.6% (standard deviation of 6.8%), with the kNN model closely trailing behind at around 90.2% (standard deviation of 6.3%). Additionally, the GNN model demonstrates superior performance in MCC, achieving a score of approximately 78.6% (standard deviation of 5.3%), with XGBoost again as the closest competitor (75.4%, standard deviation of 11.1%).

Table 6.17.: Performance comparison—Parallelism (3 features). *Best performing model

Algorithm	F_1 score	Standard deviation
LR	55.1%	5.1%
RF	71.7%	7.8%
SVM	72.8%	7.2%
kNN	67.3%	7.1%
XGBoost	77.2%	10.1%
MLP	70.4%	5.0%
GNN*	**79.6%**	**4.9%**
Algorithm	**AUC-ROC**	**Standard deviation**
LR	86.5%	5.4%
RF	80.8%	5.7%
SVM	84.5%	5.1%
kNN	90.2%	6.3%
XGBoost	86.9%	6.1%
MLP	86.6%	4.9%
GNN*	**90.6%**	**6.8%**
Algorithm	**MCC**	**Standard deviation**
LR	53.7%	6.4%
RF	70.8%	7.5%
SVM	70.5%	7.7%
kNN	65.9%	8.4%
XGBoost	75.4%	11.1%
MLP	68.0%	5.8%
GNN*	**78.6%**	**5.3%**

The experiments are repeated having selected five features per sub-process. The results are displayed in Table 6.18. The second setting, with five features per sub-process, generally performs better across all models than the first, with three features. This performance improvement indicates that the change in the number of features per sub-process has positively impacted the predictive capacity of the models. Across all metrics, the GNN consistently performed well in both settings, demonstrating high scores and relatively low standard deviations, indicating stability. In the second setting, the GNN achieved 80.4% in F_1 score with a standard deviation of 4.6%, followed by the XGBoost with 79.7% and a higher standard deviation of 8.4%. The GNN achieved 89.5% in AUC-ROC with 3.4%, while the LR appears as the second best-performing model with AUC-ROC of 88.7% and a standard deviation of 0.7%. The GNN achieved 79.1% in MCC with a standard deviation of 5.2%, followed by the XGBoost with 78.1% and a higher standard deviation of 8.6%.

Table 6.18.: Performance comparison—Parallelism (5 features). *Best performing model

Algorithm	**F_1 score**	**Standard deviation**
LR	69.2%	6.1%
RF	76.2%	6.4%
SVM	67.8%	6.7%
kNN	68.7%	3.1%
XGBoost	79.7%	8.4%
MLP	70.2%	3.5%
GNN*	**80.4%**	**4.6%**
Algorithm	**AUC-ROC**	**Standard deviation**
LR	88.7%	0.7%
RF	83.7%	3.3%
SVM	80.3%	4.3%
kNN	87.5%	3.4%
XGBoost	87.8%	5.8%
MLP	85.4%	2.0%
GNN*	**89.5%**	**3.4%**
Algorithm	**MCC**	**Standard deviation**
LR	67.2%	5.8%
RF	74.9%	7.0%
SVM	66.0%	7.9%
kNN	66.3%	3.5%
XGBoost	78.1%	8.6%
MLP	67.5%	3.8%
GNN*	**79.1%**	**5.2%**

Since the results improved from the first setting with three features per sub-process to five features, the experiments are repeated in a third setting with ten features per sub-process. Table 6.19 displays the data model comparison for the third setting. Once more, data model 2a is selected for comparison against the baselines due to its superior F_1 score.

Table 6.19.: Data models performance comparison—Parallelism (10 features)

Data model	Precision	Recall	F_1 score	AUC-ROC	MCC
1a	62.5%	71.4%	66.7%	83.6%	63.4%
1b	52.0%	92.8%	66.7%	92.3%	65.9%
2a*	**84.6%**	**78.5%**	**81.5%**	**88.6%**	**79.8%**
2b	77.8%	50.0%	60.9%	74.3%	59.6%

Table 6.20 presents the performance results of the GNN and baseline models for 10 features per sub-process for the parallelism label. All models show worse performance in the third setting. The GNN still consistently outperforms other models across all three metrics. It achieves the highest scores for F_1-score (78.9% with standard deviation of 1.8%), AUC-ROC (89.3% and standard deviation of 3.0%), and MCC (77.2% and standard deviation of 1.9%). Only the GNN model showed a lower standard deviation in the third setting compared to the second setting. The baseline models, on the contrary, presented higher standard deviations. Since the results did not improve with more features, the number of features per sub-process was not increased, and no further experiments were conducted.

Table 6.20.: Performance comparison—Parallelism (10 features). *Best performing model

Algorithm	F_1 score	Standard deviation
LR	62.2%	6.0%
RF	73.3%	10.0%
SVM	75.5%	5.9%
kNN	69.7%	6.0%
XGBoost	69.6%	13.2%
MLP	76.2%	10.5%
GNN*	**78.9%**	**1.8%**
Algorithm	**AUC-ROC**	**Standard deviation**
LR	83.0%	5.5%
RF	84.0%	6.4%
SVM	83.6%	3.3%
kNN	88.8%	3.2%
XGBoost	81.8%	9.3%
MLP	86.7%	6.2%
GNN*	**89.3%**	**3.0%**
Algorithm	**MCC**	**Standard deviation**
LR	59.0%	6.9%
RF	71.3%	10.4%
SVM	74.0%	6.3%
kNN	67.5%	6.4%
XGBoost	67.6%	13.5%
MLP	74.2%	11.5%
GNN*	**77.2%**	**1.9%**

The precision-recall curves for one random seed for the GNN and baseline models are presented in Figure 6.16 for the parallelism label. The GNN exhibits, in this case, a more evident superiority, positioning its curve in the upper-right corner of the plot while other models are situated below.

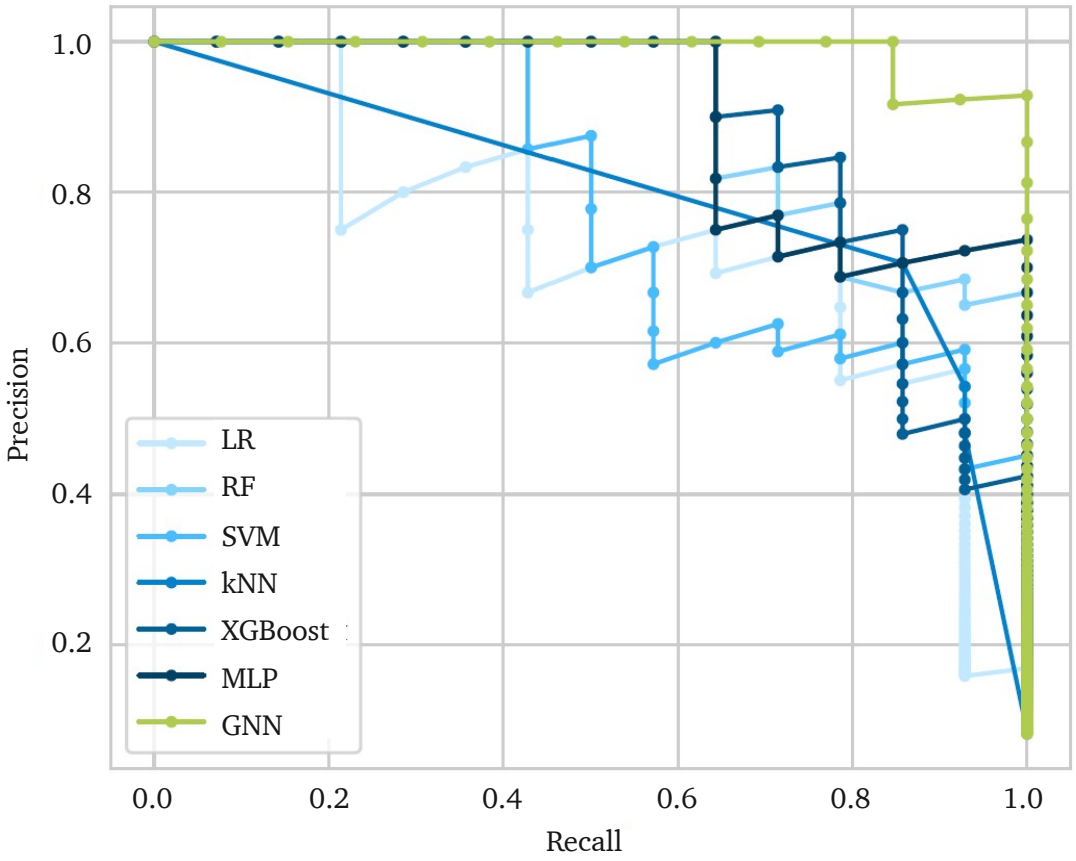


Figure 6.16.: Precision-recall curves—Parallelism

6.2.3. Discussion

The GNN outperforms the baseline models in all presented classification metrics for both quality labels. The GNN's ability to perform message-passing over the graph facilitates a contextual understanding of each node within its local and global environment. This is particularly advantageous in tasks where the importance of a data point is influenced by its relationships with neighboring points. For example, understanding how neighboring processes' quality may influence a part's quality is crucial in quality control scenarios. The GNN demonstrates effectiveness in the specific context of binary classification tasks on the provided dataset and provided labels. Its ability to capture complex relationships in graph-structured data potentially contributed to its superior performance in achieving a balanced trade-off between precision and recall (as indicated by F_1 scores and the precision-recall curves in Figures 6.15 and 6.16) and overall binary classification quality (MCC). The superiority of the GNN suggests that the graph structure inherent in the data plays a role in achieving accurate predictions. Next, each objective stated at the beginning of the section is discussed based on the achieved results.

Objective 1: impact of different graph data models The experimental results demonstrated the superiority of model 2a over other data models across all labels and experimental configurations—whether utilizing three, five, or, in the case of parallelism, ten features per sub-process. As discussed earlier, this underscores the importance of establishing directional relationships based on the intended flow of information during message passing. Opting for process nodes as the focal points for information aggregation proves more advantageous than feature nodes. In this context, 'focal points' refer to the specific locations within the graph where information is propagated and aggregated during message passing. Additionally, incorporating one-hot-encoded information into the attributes of Feature nodes yielded notable benefits. This supplementary information enhanced the representational capacity of the graphs, thereby boosting overall performance levels.

Objective 2: impact of number of features per sub-process The experiments revealed that the impact of changing the number of features per sub-process varied across different quality labels. Optimal performance for the surface roughness label was obtained with three features per sub-process, although remaining closely comparable to the performance achieved with five features: The F_1 score showed a marginal improvement of 1.3% (91.5% compared to 90.3%); the AUC-ROC remained virtually unchanged (95.5% compared to 95.3%); the MCC exhibited a slight increase of 1.7% (87.5% compared to 89.0%). Increasing the number of features to five did not yield enhanced classification metrics, suggesting that a smaller set of features is already sufficient to achieve optimal performance for the surface roughness label.

Conversely, the most favorable outcomes for the parallelism label were observed with five features per sub-process: The F_1 score recorded 79.6% in the first setting, 80.4% in the second setting, and 78.9% in the third setting; the AUC-ROC remained relatively stable at 90.6% in the first setting, 89.5% in the second setting, and 89.3% in the third setting; while the MCC exhibited values of 78.6%, 79.1%, and 77.2% respectively. Interestingly, an escalation from five to ten features per sub-process failed to improve the classification metrics. These findings underscore the dependence of feature selection on the specific quality label under consideration. Despite the label-specific differences in optimal feature counts, there appears to be a limit to the utility of adding more features, as observed in the plateauing or slight decrease in performance beyond a certain feature threshold. This conclusion highlights the importance of balancing model complexity with generalization, as overly complex models may suffer from overfitting and diminished performance on unseen data.

The differing optimal feature counts for different quality labels indicate variations in each label's complexity or underlying patterns. The results suggest the need for label-specific model tuning or feature engineering strategies to capture and characterize these nuances effectively.

Objective 3: GNN's performance compared to baselines The GNN approach consistently outperformed all other models across various classification metrics, particularly evident in its superiority for the surface roughness label. While the absolute performance gains compared to the second-best baseline model are relatively modest (approximately 1.0% in F_1 score, 1.6% in AUC-ROC, and 1.0% in MCC in the optimal setting), the GNN approach exhibited the narrowest standard deviation, indicating a more stable and consistent performance across different random seeds. This trend becomes even more pronounced when considering the parallelism label. Across the parallelism experiments, all models demonstrated inferior performance and higher standard deviations, with the GNN approach consistently maintaining the lowest standard deviation. In the optimal setting (utilizing five features per sub-process), the absolute performance gains for the parallelism label were approximately 1.0% across all classification metrics. Remarkably, the standard deviations of the GNN approach were nearly 55%, 58%, and 60% lower than those of the second-best-performing model (XGBoost) for the F_1 score, AUC-ROC, and MCC scores, respectively. The lower standard deviations of the GNN approach compared to the second-best-performing model show its superior stability in prediction across different random seeds and data subsets. This stability implies a reduced risk of unpredictable performance fluctuations, enhancing the model's trustworthiness and applicability.

While the GNN outperforms other models in this specific scenario, choosing the most suitable model should be guided by carefully considering task requirements, interpretability needs, computational resources, and generalization capabilities. The results suggest that, in scenarios where graph-structured data and complex relationships are prevalent, the GNN may be a strong candidate. Deploying the evaluated model into production for end-of-line quality control involves setting a confidence threshold, allowing automated decisions for high-confidence predictions while triggering manual checks for lower-confidence instances. Once integrated into the production workflow, the model improves operations by reducing reliance on manual checks for confidently classified products. Continuous monitoring, adaptive thresholds, and a human-in-the-loop mechanism ensure ongoing reliability. In summary, the results allow us to conclude that the proposed approach using graph representation learning and GNN for graph-level classification is promising for binary classification tasks on the provided dataset, leveraging its ability to capture complex relationships in manufacturing data. However, careful consideration of various factors and further exploration is warranted to make informed decisions about model selection and deployment.

6.3. Similarities and differences between the case studies

The two case studies, focusing on the Bosch production line performance data and the CiP-DMD, reveal both similarities and differences in their approaches and outcomes. Both studies operate within multi-stage discrete manufacturing processes. They result in the GNN approach as the top-performing model across the evaluation metrics, underscoring its efficacy in capturing complex manufacturing relationships. However, notable differences emerge between the two case studies. While Bosch centers on assembly processes, CiP-DMD's focus lies in sawing and milling operations, reflecting distinct manufacturing domains. Additionally, Bosch contains anonymized features, whereas CiP-DMD utilizes non-anonymized features. Furthermore, the labeling schemes differ between the two studies, with Bosch employing a binary pass/fail criterion, while CiP-DMD utilizes quality measurement labels, suggesting variations in the granularity of quality assessment. Data formats also vary, with Bosch utilizing tabular data and CiP-DMD working with time series data, necessitating different feature extraction approaches. In terms of dataset characteristics, Bosch is a larger dataset with 11,592 samples compared to CiP-DMD's 824 samples. CiP-DMD also exhibits a higher proportion of nonconforming samples, with 22.7% for surface roughness and 8.4% for parallelism, compared to Bosch's 0.74%, which impacts model performance and generalization. Most significantly, while both studies identify the GNN as the top-performing model, there are major differences in performance metrics. Case study 1 achieves a 36% F_1 score, whereas case study 2 achieves significantly higher F_1 scores of 91.5% for surface roughness and 80.4% for parallelism. These similarities and differences are summarized in Table 6.21.

Table 6.21.: Similarities and differences between case studies. SR = surface roughness, P = parallelism

Case study	**Case study 1: Bosch**	**Case study 2: CiP-DMD**
Similarities	Multi-stage discrete manufacturing process	Multi-stage discrete manufacturing process
	GNN top-performer across all metrics	GNN top-performer across all metrics
Differences	Assembly	Sawing and milling
	Anonymized features	Non-anonymized features
	Label pass/fail	Quality measurements
	Tabular data	Time series data
	11,592 samples	824 samples
	0.74% of NOK samples	22.7% of NOK samples (SR)

Table 6.21 (continued)

Case study	Case study 1: Bosch	Case study 2: CiP-DMD
		8.4% of NOK samples (P)
	Low performance of all models	Higher performance of all models
	36% F_1 score	91.5% F_1 score (SR)
		80.4% F_1 score (P)

The study evaluating the proposed approach compared to baseline models, as developed for both case studies in this thesis, reveals several strengths. Firstly, it demonstrates the robust performance of the GNN approach, which consistently outperforms baseline models across all evaluation metrics. The superior performance indicates the efficacy of the GNN in capturing the relationships present in manufacturing datasets, particularly within multi-stage discrete manufacturing processes. Moreover, the study highlights the generalizability of the GNN approach across diverse manufacturing scenarios. Despite differences in manufacturing domains, data characteristics, and labeling schemes between the two case studies, the GNN emerges as the top-performing model in both contexts, suggesting its potential applicability in various industrial settings. The study also shows that the proposed approach can effectively handle tabular data, as seen in the Bosch case study, as well as time series data, as observed in the CiP-DMD case study. This versatility highlights the proposed approach's capability to accommodate different data structures and extraction methods, enhancing its utility in analyzing manufacturing datasets of varying complexities. Additionally, by comparing performance metrics across different case studies, the study provides valuable insights into the strengths and limitations of the GNN approach relative to baseline models. This comparison enables informed decisions regarding model selection and deployment based on specific application requirements and dataset characteristics. Finally, the study identifies commonalities in the effectiveness of the GNN approach despite differences in manufacturing processes and dataset characteristics between the two case studies. The results suggest that the GNN may possess inherent capabilities to capture process-specific nuances and variations, contributing to its superior performance across diverse manufacturing environments. These findings accentuate the potential of the GNN approach as a versatile and effective tool for analyzing manufacturing data and performing classification tasks.

6.4. Practical recommendations

The practical recommendations presented in this section are intended for those aiming to implement the proposed graph representation learning approach in further case studies and datasets. These recommendations come from insights from the presented case studies and are categorized into three main areas: the importance of the data model, the impact of hyperparameters for the GNN architecture, and the training process.

Importance of the data model The experiments show the critical role of the initial step of the proposed approach (I Identify, illustrated in Figure 5.1) in modeling the interrelationships between variables and steps within the manufacturing process. Notably, the most effective data models share a common feature: They represent relationships directed from features to stations or processes. In directed graphs, like in the cases presented in this thesis, node attributes gather distinct representations during message passing based on their incoming and outgoing connections, capturing different aspects of their roles in the graph structure. This commonality in the results indicates that having the MPS as central nodes where information is aggregated is beneficial. Furthermore, the experiments highlight the necessity of evaluating whether one-hot-encoded

representations of feature attributes are indispensable for each specific case, as this additional information did not uniformly enhance performance across all case studies.

Importance of hyperparameters for the GNN architecture The experiments in this thesis showed that the GraphSAGE layer performed better than the other two types of layers. In both case studies, the GraphSAGE layer was selected during hyperparameter tuning in the cross-validation process and was used in the best-performing GNN models. The readout function 'sum', which performs a sum of all nodes in the graph, performed better than alternative readout functions such as 'min', 'max', and 'mean'. The superior performance could be because the 'sum' readout function aggregates information from all nodes equally, which might be beneficial if each node contributes essential information to the overall graph representation. In contrast, other readout functions like 'mean' or 'max' might dilute or prioritize certain node features over others, leading to loss of crucial information. The choice of readout function interacts with the overall model architecture and optimization process. The 'sum' readout function might lead to better optimization or more stable gradients during training compared to other functions, resulting in improved performance overall.

Importance of hyperparameters for the training process To achieve optimal performance with GNN models, two hyperparameters, namely the number of epochs and batch size, emerged as particularly influential. Fine-tuning the number of epochs was crucial for mitigating overfitting, with lower epoch counts yielding the most favorable results across both case studies. In both cases (even with a drastically different number of samples in each case), the number of epochs with values between five and 15 led to the best results. Regarding the batch size, a larger batch size did not always lead to improved performance and reduced overfitting. In both case studies, smaller batch sizes performed better than larger ones.

Digression: Practical recommendations for the deployment in production In the first case study that used the Bosch production line performance data, the overall F_1 scores across all models, including the GNN, are low, with the GNN emerging as the top performer with a modest 36% F_1 score. The low F_1 scores suggest that the GNN approach, while the best among the models evaluated, still struggles to differentiate between pass and fail products effectively. Case study 1 serves as a valuable exploration into comparing the efficacy of the GNN against baseline approaches, shedding light on the potential promise of the proposed methodology. However, in this case, it does not provide conclusive evidence that an ML approach can entirely or partially substitute end-of-line quality control measures.

The second case study involving the CiP-DMD dataset demonstrates higher F_1 score performance, for the quality labels surface roughness and parallelism. The GNN emerges as the top performer with F_1 scores of 91.5% for surface roughness and 80.4% for parallelism. These high F_1 scores indicate the GNN's ability to learn and find patterns in sawing and milling data for predicting the surface roughness and, to a lower extent, parallelism labels studied in this case. The GNN model can potentially help reduce the number of manually inspected parts for both labels. The classification threshold after the sigmoid function can be adjusted to reduce the number of false negatives (reach a recall of 1) and thus prevent defective parts from being classified as quality conform (as indicated in Figures 6.15 and 6.16). The trained GNN model can then be deployed to assist in inspecting parts identified as non-quality conforming, thus reducing the total number of parts being manually inspected and thus reducing costs.

Acknowledging the limitations and uncertainties inherent in deploying ML models, including GNNs, in real-world manufacturing applications is essential. Challenges such as data quality, model generalization, and interpretability necessitate careful consideration to ensure the viability and efficacy of GNN-based solutions. Section 7.2 discusses these aspects, where the suggested future work is presented.

7. Conclusion

The experimental results affirm the effectiveness of the proposed GNN approach compared to baseline ML algorithms for the tasks of quality classification in manufacturing processes. The GNN model consistently outperforms the baseline models, highlighting its potential to enhance quality control in manufacturing environments. The superior performance of the GNN approach across diverse manufacturing processes underscores its generalizability in different operational contexts. Notably, the first case study focusing on assembly processes at Bosch and the second case study involving sawing and milling operations with the CiP-DMD dataset represent distinct manufacturing domains with unique characteristics and challenges. Despite these differences, the GNN consistently outperforms baseline ML algorithms in both case studies. This superior performance indicates that the GNN's ability to model feature dependencies within a graph framework is advantageous across various manufacturing processes, regardless of the specific operations and data types (tabular or time series) involved.

Answer to the research question The comparison of the proposed GNN approach with baseline ML algorithms across both case studies consistently demonstrates the superior performance of the GNN models. In the first case study focused on Bosch production line performance data, the GNN emerges as the top performer. Although the performance remains suboptimal in absolute terms, the GNN still outperforms the identified baseline ML algorithms, indicating its superiority in classifying quality labels within the assembly processes studied. Similarly, in the second case study involving the CiP-DMD dataset, the GNN approach exhibits higher performance levels than the baseline ML algorithms. With F_1 scores of 91.5% for surface roughness and 80.4% for parallelism, the GNN achieves superior results in predicting these quality parameters within the sawing and milling operations.

Evaluation of the hypothesis The hypothesis that the proposed GNN approach outperforms the baseline ML algorithms is consistently supported by the findings from both case studies.

7.1. Summary

This thesis introduces an innovative approach for predictive quality in multi-stage discrete manufacturing. Specifically, it explores a graph representation learning method for classifying products as quality-conforming or nonconforming based on data gathered from manufacturing. The rationale behind investigating graph representation learning lies in conceptualizing a multi-stage manufacturing process as a connected network, where each stage's input is derived from the output of its preceding stage. This interconnectedness can be effectively depicted through a heterogeneous graph, where nodes represent real-world entities and edges denote their relationships. The objective is to assess the efficacy of a technology proven successful in other scientific domains, such as biochemistry, to ascertain whether it can outperform ML algorithms currently employed in manufacturing for predictive quality solutions.

The theoretical foundation, outlined in Chapter 2, explains the relevant concepts and theories. It delineates the characteristics of a multi-stage discrete manufacturing process and investigates quality management,

particularly quality control for such processes. A brief historical overview of quality management, alongside the integration of recent ML technologies into this domain, precedes an introduction to predictive quality. The chapter then explains ML concepts, emphasizing classification and classification metrics, given the proposed approach's tailored focus on binary classification tasks. Lastly, it introduces graph theory and ML on graph data, laying the groundwork to comprehend GNNs and how the proposed approach leverages this data representation for graph classification.

Chapter 3, titled 'Existing approaches and required action', presents findings from a systematic literature review to identify research gaps and ML algorithms utilized in manufacturing literature for predictive quality solutions. It highlights the challenges encountered by existing methods—such as low model performance, extensive feature engineering requirements, and the complexity of multi-stage processes—and establishes baselines for contrasting the proposed approach, including LR, RF, SVM, kNN, XGBoost, and MLP.

Chapter 4 delineates the research design, formulating the research problem, the research question, and the hypothesis. It also introduces CRISP-ML(Q), a process model guiding the proposed method and the subsequent case studies.

Chapters 5 and 6 constitute the main scientific contribution of this thesis. Chapter 5 details the 'Graph representation learning for predictive quality in discrete manufacturing' approach, comprising six steps: identification of minimal processing steps and variables, feature extraction and selection, graph dataset creation, GNN model training, and evaluation. The evaluation framework, incorporating cross-validation, assesses the proposed approach alongside baselines. The chapter provides a comprehensive overview of the GNN architecture and hyperparameters employed in the case studies for graph classification.

Chapter 6 validates the proposed approach through two case studies—one utilizing the Bosch production line performance dataset and the other the CiP-DMD dataset—alongside experiments comparing the proposed method with baseline ML algorithms. Each case study begins with an overview of the manufacturing process and available data, followed by the CRISP-ML(Q) phases and the evaluation of the proposed method. In both studies, the GNN approach demonstrated superior performance in F_1 scores, and MCC, with lower standard deviations than baseline approaches. The chapter concludes by contrasting case studies and offering practical recommendations for implementing the proposed method in further manufacturing scenarios.

7.2. Limitations and future work

Limitations The thesis presents a promising approach for developing predictive quality applications in multi-stage discrete manufacturing. However, it is necessary to acknowledge its limitations, which can be categorized into three primary areas: data, methodological, and scope limitations.

- Data limitations: The case studies conducted to validate the proposed approach are limited to specific datasets, namely the Bosch production line performance and the CiP-DMD datasets. These datasets primarily capture data from assembly processes (Bosch) and sawing and milling processes (CiP-DMD). Expanding the validation to include additional multi-stage processes, such as those from the automobile industry and semiconductor manufacturing, could provide further insights. Furthermore, the case studies in this thesis were limited to tabular and time series data. It would be helpful to assess how the proposed approach could be applied to additional data formats, such as image data.

- Methodological limitations: The thesis evaluates a certain number of data models and their corresponding representations of multi-stage processes. Alternative data models could capture different aspects of the production line, improving representation expressiveness and classifier performance. Additionally, the selection of GNN layers and their combinations for defining the model architecture is constrained to the choices outlined in Chapters 5 and 6.

- Scope limitations: The investigation of alternative methods for introducing bias to the ML models concerning feature interdependency was outside the scope of this thesis. Exploring how process interdependencies can be represented and compared to the graph representation learning approach would be an interesting analysis for future research.

Future work Identified from the results and limitations presented, five research directions for future work are highlighted below:

- Improving GNN architecture and hyperparameter optimization: Future research could explore strategies for architectural refinement and hyperparameter optimization to improve the proposed GNN architecture. This may involve experimenting with different GNN architectures, such as other graph layer types, combination of different layer types in the same model, and verifying their suitability for predictive quality tasks in discrete manufacturing.
- Investigating alternative data models and their impact on performance: Proposing and testing alternative graph data models could be an interesting direction for future research. This includes exploring different ways of representing entities within the manufacturing process and defining node classes, relationship types, and attributes. Researchers could investigate variations in graph construction methodologies, such as node aggregation techniques, edge weight assignments, or graph partitioning strategies, to capture different aspects of the manufacturing process.
- Investigating alternative feature selection methods and their impact on performance: Future research could explore further feature selection methods, such as additional filter methods, and extend the scope to include graph features, not only limited to those extracted from the manufacturing process but also descriptive of the graph characteristics. Comparative evaluations of different feature selection methods, coupled with sensitivity analyses to assess feature importance and relevance, could be used to assess the impact of feature selection on model performance and generalizability.
- Diversifying dataset exploration and validation: Future research should explore applying the proposed approach across diverse manufacturing datasets representing various industry sectors and production processes to broaden the scope of empirical validation. Systematic experimentation with diverse datasets can provide insights into the generalizability and transferability of the proposed approach across heterogeneous manufacturing environments.

A. Manufacturing process cylinder bottom—CiP case study

Table A.1.: Manufacturing process cylinder bottom

Nr.	Main process	Machine and tools description
1	Cut raw material onto 20 mm blank.	Saw (Kasto SBA A2)
1	Scan QR code.	Saw (Kasto SBA A2)
2	Quality inspection at the scale.	Scale
2	Attach sticker to blank.	Saw (Kasto SBA A2) Scanner
3	Transport blank to the milling machine.	Transport
3	Scan QR code and remove sticker from blank.	Milling machine (DMC 50H) Scanner
3	Clamp blank in vice on side 1.	Milling machine (DMC 50H) Tower 1 Torque wrench
3	Clamp semi-finished part in vice on side 2.	Milling machine (DMC 50H) Tower 1 Torque wrench
3	Pallet change: Changing the tower from setup to machining position.	Milling machine (DMC 50H)
3	Program selection: Select program for tower 1.	Milling machine (DMC 50H)
3	Program start: Start program for tower 1.	Milling machine (DMC 50H)
3	Face milling: Face mill the surface of the blank.	Milling machine (DMC 50H) Cutter head D50
3	Roughing the outer contour: Producing the outer contour with an oversize.	Milling machine (DMC 50H) End mill D10
3	Smoothing the outer contour: Finishing the outer contour.	Milling machine (DMC 50H) End mill D10
3	Milling the groove: Milling the lateral groove.	Milling machine (DMC 50H) End mill D6
3	Step drilling: Drill mounting holes.	Milling machine (DMC 50H) Step drill D6.6
3	Deburring of the outer contour and step bore.	Milling machine (DMC 50H) Deburrer D10
3	Rotate tower 1 by 90°.	Milling machine (DMC 50H)
3	Drill core hole for G1/8” thread.	Milling machine (DMC 50H) Drill D8,6

Table A.1 (continued)

Nr.	Process	Machine and tools description
3	Countersink hole D8,6.	Milling machine (DMC 50H) Deburrer D10
3	Drill compressed air connection hole D5.	Milling machine (DMC 50H) Drill D5
3	Milling of the connection thread G1/8”.	Milling machine (DMC 50H) Thread milling cutter G1/8”
3	Rotate tower 1 by 90°.	Milling machine (DMC 50H)
3	Milling the circular stud on side 2.	Milling machine (DMC 50H) Cutter head D50
3	Face milling of the component on side 2.	Milling machine (DMC 50H) Cutter head D50
3	Milling the circular pocket on side 2.	Milling machine (DMC 50H) End mill D10
3	Deburring of the outer contour and through hole.	Milling machine (DMC 50H) Deburrer D10
3	Spindle the ring groove for the sealing ring.	Milling machine (DMC 50H) Custom tool D40
3	Pallet change: Change of the tower from machining to setup position.	Milling machine (DMC 50H)
3	Removal of the finished part from side 2.	Milling machine (DMC 50H) Torque wrench
3	Scan QR code.	Milling machine (DMC 50H) Scanner
4	Transport part to washing station.	Kärcher Bio M1 (washing station)
4	Cleaning: Clean part from cooling lubricant and swarf residues.	Kärcher Bio M1 (washing station)
5	Transport part to measuring station.	Measuring station
5	Scan QR code and temporarily remove sticker with QR code.	Measuring station Scanner
5	Quality control: Check quality characteristics.	Measuring station
5	Attach sticker with QR code.	Measuring station

B. Selected features—CiP case study

Table B.1 lists the sub-processes involved in manufacturing the cylinder bottom recorded in the CiP-DMD dataset. Table B.2 presents the selected features and their respective configurations for each label and sub-process. Specifically, three and five features were selected for the surface roughness label, while for the parallelism label, three, five, and ten features were chosen, as elaborated in Section 6.2. Further details regarding each feature extracted with the tsfresh library can be found in the comprehensive documentation provided by [105].

Table B.1.: Processes cylinder bottom

Process ID	Sub-process
1	Sawing
2	Face cutting
3	Outer contour roughing and finishing
4	Groove side milling
5	Step boring
6	End burring and external contour boring
7	Lateral drilling
8	Drilling sinking
9	Drilling
10	Thread milling
11	Face cutting
12	Circular pocket milling
13	Deburring
14	Ring groove milling

Table B.2.: Selected features and tsfresh configuration for each label and process

Label	Process	Feature configuration
Surface roughness	1	CutCounter cwt_coefficients: (coeff: 6, w: 20, widths: (2, 5, 10, 20))
		CutCounter cwt_coefficients: (coeff: 9, w: 20, widths: (2, 5, 10, 20))
		CutCounter cwt_coefficients: (coeff: 8, w: 20, widths: (2, 5, 10, 20))
		CutCounter cwt_coefficients: (coeff: 7, w: 20, widths: (2, 5, 10, 20))
		CutCounter cwt_coefficients: (coeff: 10, w: 20, widths: (2, 5, 10, 20))
	2	Acc_x ar_coefficient: (coeff: 8, k: 10)
		Acc_y permutation_entropy: (dimension: 3, tau: 1)
		Acc_y energy_ratio_by_chunks: (num_segments: 10, segment_focus: 7)

Table B.2 (continued)

Label	Process	Feature configuration
		Acc_y friedrich_coefficients: (coeff: 0, m: 3, r: 30)
		Acc_z energy_ratio_by_chunks: (num_segments: 10, segment_focus: 7)
	3	Acc_y fourier_entropy: (bins: 100)
		Acc_y benford_correlation
		Acc_y median
		Acc_y ratio_beyond_r_sigma: (r: 7)
		Acc_z number_peaks: (n: 5)
	4	Acc_y change_quantiles: (f_agg: mean, isabs: False, qh: 0.6, ql: 0.0)
		Acc_y change_quantiles: (f_agg: mean, isabs: False, qh: 0.4, ql: 0.0)
		Acc_y change_quantiles: (f_agg: mean, isabs: False, qh: 0.6, ql: 0.2)
		Acc_z time_reversal_asymmetry_statistic: (lag: 1)
		Acc_z ratio_beyond_r_sigma: (r: 0.5)
	5	Acc_y fourier_entropy: (bins: 5)
		Acc_y fourier_entropy: (bins: 3)
		Acc_y fourier_entropy: (bins: 2)
		Acc_y fourier_entropy: (bins: 10)
		Acc_x permutation_entropy: (dimension: 7, tau: 1)
	6	Acc_y ar_coefficient: (coeff: 10, k: 10)
		Acc_z agg_autocorrelation: (f_agg: median, maxlag: 40)
		Acc_z ar_coefficient: (coeff: 9, k: 10)
		Acc_z ar_coefficient: (coeff: 7, k: 10)
		Acc_z partial_autocorrelation: (lag: 2)
	7	Acc_x permutation_entropy: (dimension: 7, tau: 1)
		Acc_x permutation_entropy: (dimension: 6, tau: 1)
		Acc_z partial_autocorrelation: (lag: 2)
		Acc_z autocorrelation: (lag: 5)
		Acc_y ar_coefficient: (coeff: 4, k: 10)
	8	Acc_z partial_autocorrelation: (lag: 2)
		Acc_z autocorrelation: (lag: 5)
		Acc_z ar_coefficient: (coeff: 3, k: 10)
		Acc_x fourier_entropy: (bins: 100)
		Acc_x permutation_entropy: (dimension: 7, tau: 1)
	9	Acc_x permutation_entropy: (dimension: 3, tau: 1)
		Acc_x ar_coefficient: (coeff: 4, k: 10)
		Acc_x change_quantiles: (f_agg: var, isabs: True, qh: 0.8, ql: 0.6)
		Acc_z agg_autocorrelation: (f_agg: var, maxlag: 40)
		Acc_z quantile: (q: 0.7)
	10	Acc_z partial_autocorrelation: (lag: 2)
		Acc_z partial_autocorrelation: (lag: 4)
		Acc_z ar_coefficient: (coeff: 3, k: 10)
		Acc_x ar_coefficient: (coeff: 1, k: 10)
		Acc_x partial_autocorrelation: (lag: 3)
	11	Acc_x ratio_beyond_r_sigma: (r: 0.5)

Table B.2 (continued)

Label	Process	Feature configuration
		Acc_x spkt_welch_density: (coeff: 8)
		Acc_z spkt_welch_density: (coeff: 5)
		Acc_y augmented_dickey_fuller: (attr: pvalue, autolag: AIC)
		Acc_y augmented_dickey_fuller: (attr: teststat, autolag: AIC)
	12	Acc_x maximum
		Acc_x mean_n_absolute_max: (number_of_maxima: 7)
		Acc_x change_quantiles: (f_agg: var, isabs: False, qh: 1.0, ql: 0.0)
		Acc_x cid_ce: (normalize: False)
		Acc_z number_crossing_m: (m: 1)
	13	Acc_z ar_coefficient: (coeff: 2, k: 10)
		Acc_z ar_coefficient: (coeff: 9, k: 10)
		Acc_z ar_coefficient: (coeff: 1, k: 10)
		Acc_z ar_coefficient: (coeff: 4, k: 10)
		Acc_y change_quantiles: (f_agg: var, isabs: True, qh: 1.0, ql: 0.6)
	14	Acc_x autocorrelation: (lag: 5)
		Acc_x change_quantiles: (f_agg: var, isabs: True, qh: 0.4, ql: 0.2)
		Acc_x change_quantiles: (f_agg: var, isabs: True, qh: 0.8, ql: 0.2)
		Acc_x change_quantiles: (f_agg: var, isabs: True, qh: 0.6, ql: 0.2)
		Acc_x quantile: (q: 0.2)
Parallelism	1	Vib01.RMS fft_coefficient: (attr: abs, coeff: 74)
		P_Vorschub fft_coefficient: (attr: angle, coeff: 75)
		TData.T1 fft_coefficient: (attr: real, coeff: 75)
		Vib02.RMS fft_coefficient: (attr: real, coeff: 77)
		CutCounter: ar_coefficient: (coeff: 10, k: 10)
		Vib01.Skewness fft_coefficient: (attr: imag, coeff: 40)
		HebenAktiv agg_linear_trend: (attr: stderr, chunk_len: 50, f_agg: min)
		HebenAktiv agg_linear_trend: (attr: rvalue, chunk_len: 50, f_agg: min)
		HebenAktiv agg_linear_trend: (attr: intercept, chunk_len: 50, f_agg: min)
		HebenAktiv agg_linear_trend: (attr: slope, chunk_len: 50, f_agg: min)
	2	Acc_z fourier_entropy: (bins: 3)
		Acc_z fourier_entropy: (bins: 2)
		Acc_z fourier_entropy: (bins: 5)
		Acc_z autocorrelation: (lag: 5)
		Acc_z ratio_beyond_r_sigma: (r: 7)
		Acc_x count_below: (t: 0)
		Acc_x count_above: (t: 0)
		Acc_x c3: (lag: 1)
		Acc_x max_langevin_fixed_point: (m: 3, r: 30)
		Acc_y fourier_entropy: (bins: 2)
	3	Acc_z friedrich_coefficients: (coeff: 2, m: 3, r: 30)
		Acc_z number_peaks: (n: 10)
		Acc_z fourier_entropy: (bins: 100)
		Acc_z autocorrelation: (lag: 2)

Table B.2 (continued)

Label	Process	Feature configuration
		Acc_z permutation_entropy: (dimension: 3, tau: 1)
		Acc_z partial_autocorrelation: (lag: 1)
		Acc_x permutation_entropy: (dimension: 5, tau: 1)
		Acc_x permutation_entropy: (dimension: 4, tau: 1)
		Acc_x number_peaks: (n: 10)
		Acc_x autocorrelation: (lag: 3)
	4	Acc_y change_quantiles: (f_agg: mean, isabs: False, qh: 1.0, ql: 0.8)
		Acc_y change_quantiles: (f_agg: mean, isabs: False, qh: 1.0, ql: 0.6)
		Acc_y change_quantiles: (f_agg: mean, isabs: False, qh: 0.8, ql: 0.0)
		Acc_z agg_autocorrelation: (f_agg: median, maxlag: 40)
		Acc_z agg_linear_trend: (attr: stderr, chunk_len: 50, f_agg: var)
		Acc_z agg_linear_trend: (attr: stderr, chunk_len: 10, f_agg: var)
		Acc_z agg_linear_trend: (attr: stderr, chunk_len: 10, f_agg: min)
		Acc_z agg_linear_trend: (attr: stderr, chunk_len: 10, f_agg: max)
		Acc_z agg_linear_trend: (attr: stderr, chunk_len: 50, f_agg: max)
		Acc_z change_quantiles: (f_agg: var, isabs: True, qh: 1.0, ql: 0.0)
	5	Acc_x change_quantiles: (f_agg: var, isabs: False, qh: 0.6, ql: 0.0)
		Acc_x change_quantiles: (f_agg: var, isabs: True, qh: 0.6, ql: 0.0)
		Acc_x change_quantiles: (f_agg: var, isabs: False, qh: 0.4, ql: 0.0)
		Acc_x change_quantiles: (f_agg: var, isabs: False, qh: 0.2, ql: 0.0)
		Acc_x change_quantiles: (f_agg: mean, isabs: True, qh: 0.2, ql: 0.0)
		Acc_x change_quantiles: (f_agg: var, isabs: True, qh: 0.4, ql: 0.0)
		Acc_x change_quantiles: (f_agg: mean, isabs: True, qh: 0.4, ql: 0.0)
		Acc_x change_quantiles: (f_agg: var, isabs: True, qh: 0.2, ql: 0.0)
		Acc_x absolute_maximum
		Acc_x mean_n_absolute_max: (number_of_maxima: 7)
	6	Acc_z autocorrelation: (lag: 2)
		Acc_z autocorrelation: (lag: 4)
		Acc_z autocorrelation: (lag: 6)
		Acc_z fourier_entropy: (bins: 100)
		Acc_z number_peaks: (n: 3)
		Acc_z number_peaks: (n: 5)
		Acc_z permutation_entropy: (dimension: 7, tau: 1)
		Acc_z permutation_entropy: (dimension: 6, tau: 1)
		Acc_z number_cwt_peaks: (n: 1)
		Acc_y ar_coefficient: (coeff: 4, k: 10)
	7	Acc_y partial_autocorrelation: (lag: 3)
		Acc_y absolute_sum_of_changes
		Acc_y cid_ce: (normalize: False)
		Acc_y change_quantiles: (f_agg: mean, isabs: True, qh: 1.0, ql: 0.0)
		Acc_y change_quantiles: (f_agg: var, isabs: True, qh: 1.0, ql: 0.0)
		Acc_y mean_abs_change
		Acc_y number_crossing_m: (m: 0)

Table B.2 (continued)

Label	Process	Feature configuration
		Acc_x ar_coefficient: (coeff: 10, k: 10)
		Acc_x ar_coefficient: (coeff: 9, k: 10)
		Acc_x absolute_sum_of_changes
	8	Acc_x ar_coefficient: (coeff: 10, k: 10)
		Acc_x fourier_entropy: (bins: 100)
		Acc_x autocorrelation: (lag: 8)
		Acc_x autocorrelation: (lag: 1)
		Acc_x partial_autocorrelation: (lag: 1)
		Acc_y fourier_entropy: (bins: 3)
		Acc_y fourier_entropy: (bins: 2)
		Acc_y autocorrelation: (lag: 4)
		Acc_y change_quantiles: (f_agg: var, isabs: False, qh: 1.0, ql: 0.6)
		Acc_y change_quantiles: (f_agg: var, isabs: True, qh: 1.0, ql: 0.2)
	9	Acc_y change_quantiles: (f_agg: mean, isabs: True, qh: 1.0, ql: 0.8)
		Acc_y change_quantiles: (f_agg: var, isabs: False, qh: 1.0, ql: 0.8)
		Acc_y change_quantiles: (f_agg: var, isabs: True, qh: 1.0, ql: 0.8)
		Acc_y change_quantiles: (f_agg: var, isabs: False, qh: 0.2, ql: 0.0)
		Acc_y change_quantiles: (f_agg: var, isabs: True, qh: 0.2, ql: 0.0)
		Acc_y permutation_entropy: (dimension: 3, tau: 1)
		Acc_y permutation_entropy: (dimension: 4, tau: 1)
		Acc_x partial_autocorrelation: (lag: 5)
		Acc_x ar_coefficient: (coeff: 5, k: 10)
		Acc_x cid_ce: (normalize: True)
	10	Acc_x permutation_entropy: (dimension: 7, tau: 1)
		Acc_x permutation_entropy: (dimension: 6, tau: 1)
		Acc_x permutation_entropy: (dimension: 5, tau: 1)
		Acc_x permutation_entropy: (dimension: 4, tau: 1)
		Acc_x permutation_entropy: (dimension: 3, tau: 1)
		Acc_x partial_autocorrelation: (lag: 5)
		Acc_x ar_coefficient: (coeff: 5, k: 10)
		Acc_x cid_ce: (normalize: False)
		Acc_z autocorrelation: (lag: 8)
		Acc_z autocorrelation: (lag: 6)
	11	Acc_y ratio_beyond_r_sigma: (r: 7)
		Acc_y longest_strike_below_mean
		Acc_y skewness
		Acc_y energy_ratio_by_chunks: (num_segments: 10, segment_focus: 2)
		Acc_y fft_coefficient: (attr: abs, coeff: 86)
		Acc_x fourier_entropy: (bins: 2)
		Acc_x skewness
		Acc_x permutation_entropy: (dimension: 5, tau: 1)
		Acc_x permutation_entropy: (dimension: 7, tau: 1)
		Acc_x permutation_entropy: (dimension: 6, tau: 1)

Table B.2 (continued)

Label	Process	Feature configuration
	12	Acc_x energy_ratio_by_chunks: (num_segments: 10, segment_focus: 5)
		Acc_x energy_ratio_by_chunks: (num_segments: 10, segment_focus: 1)
		Acc_x autocorrelation: (lag: 5)
		Acc_x ratio_beyond_r_sigma: (r: 0.5)
		Acc_x index_mass_quantile: (q: 0.2)
		Acc_x index_mass_quantile: (q: 0.1)
		Acc_x agg_linear_trend: (attr: stderr, chunk_len: 50, f_agg: var)
		Acc_z time_reversal_asymmetry_statistic: (lag: 2)
		Acc_z change_quantiles: (f_agg: mean, isabs: False, qh: 0.8, ql: 0.2)
		Acc_z change_quantiles: (f_agg: mean, isabs: False, qh: 1.0, ql: 0.4)
	13	Acc_y ratio_beyond_r_sigma: (r: 5)
		Acc_y change_quantiles: (f_agg: var, isabs: True, qh: 0.4, ql: 0.0)
		Acc_y change_quantiles: (f_agg: var, isabs: False, qh: 0.4, ql: 0.0)
		Acc_y change_quantiles: (f_agg: var, isabs: True, qh: 1.0, ql: 0.2)
		Acc_y percentage_of_reoccurring_values_to_all_values
		Acc_y fourier_entropy: (bins: 100)
		Acc_y energy_ratio_by_chunks: (num_segments: 10, segment_focus: 9)
		Acc_x ar_coefficient: (coeff: 4, k: 10)
		Acc_x ar_coefficient: (coeff: 9, k: 10)
		Acc_x ar_coefficient: (coeff: 5, k: 10)
	14	Acc_x ar_coefficient: (coeff: 5, k: 10)
		Acc_x ar_coefficient: (coeff: 1, k: 10)
		Acc_x ar_coefficient: (coeff: 4, k: 10)
		Acc_z ar_coefficient: (coeff: 1, k: 10)
		Acc_z agg_autocorrelation: (f_agg: median, maxlag: 40)
		Acc_z agg_autocorrelation: (f_agg: mean, maxlag: 40)
		Acc_z partial_autocorrelation: (lag: 1)
		Acc_z autocorrelation: (lag: 1)
		Acc_z autocorrelation: (lag: 8)
		Acc_y change_quantiles: (f_agg: mean, isabs: True, qh: 0.4, ql: 0.2)

C. Case studies performance evaluation

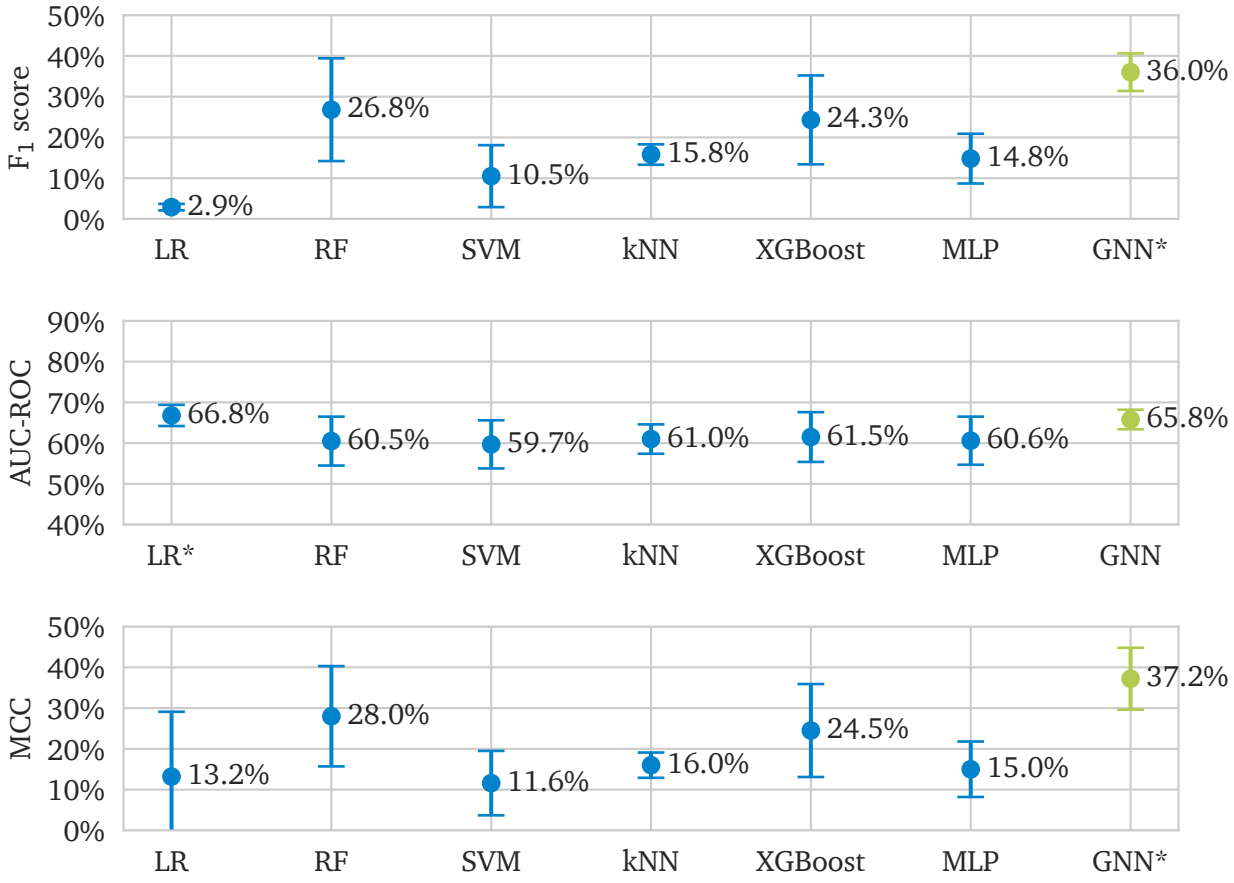


Figure C.1.: Performance comparison—Case study 1. *Best performing model

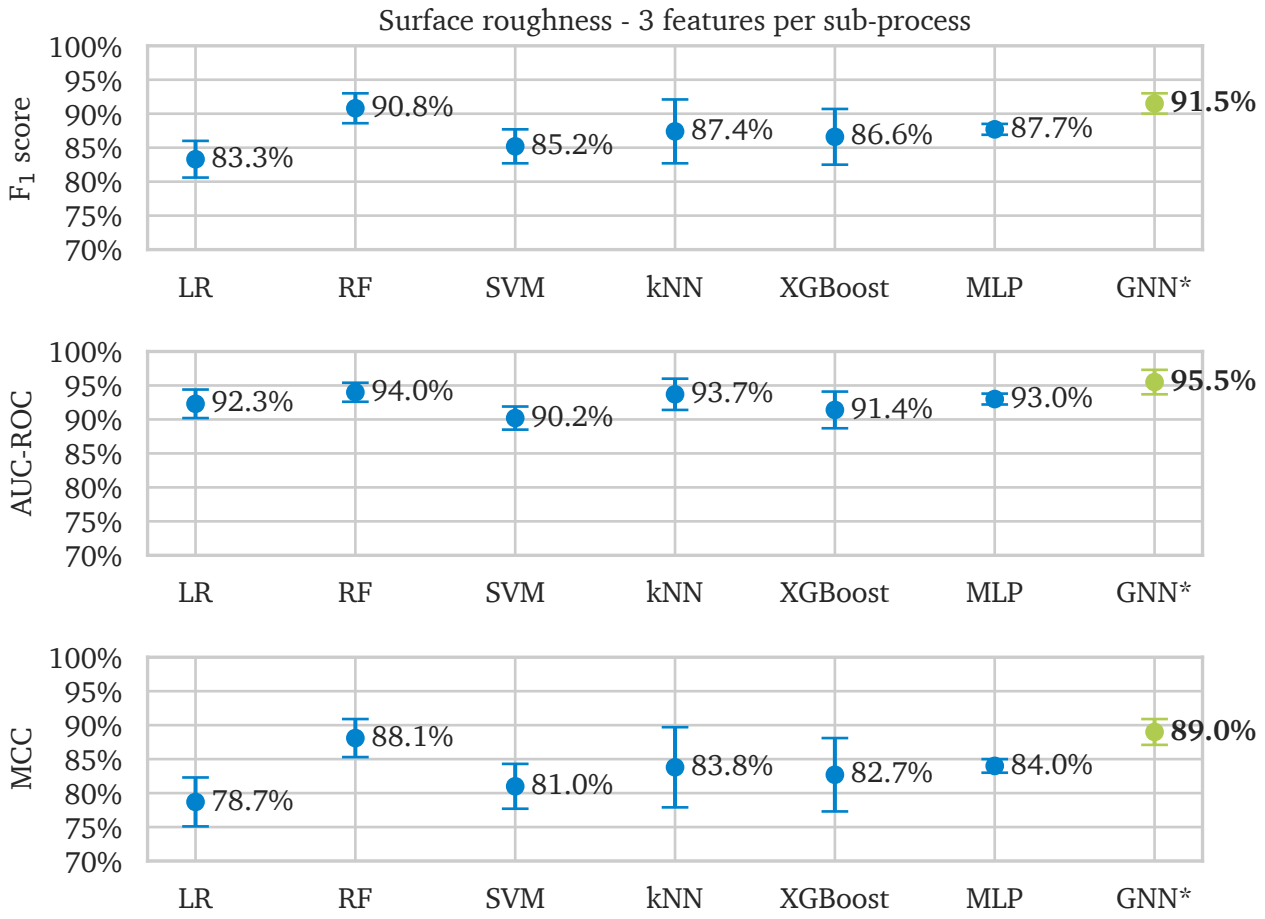


Figure C.2.: Performance comparison—Case study 2—Surface roughness (3 features). *Best performing model

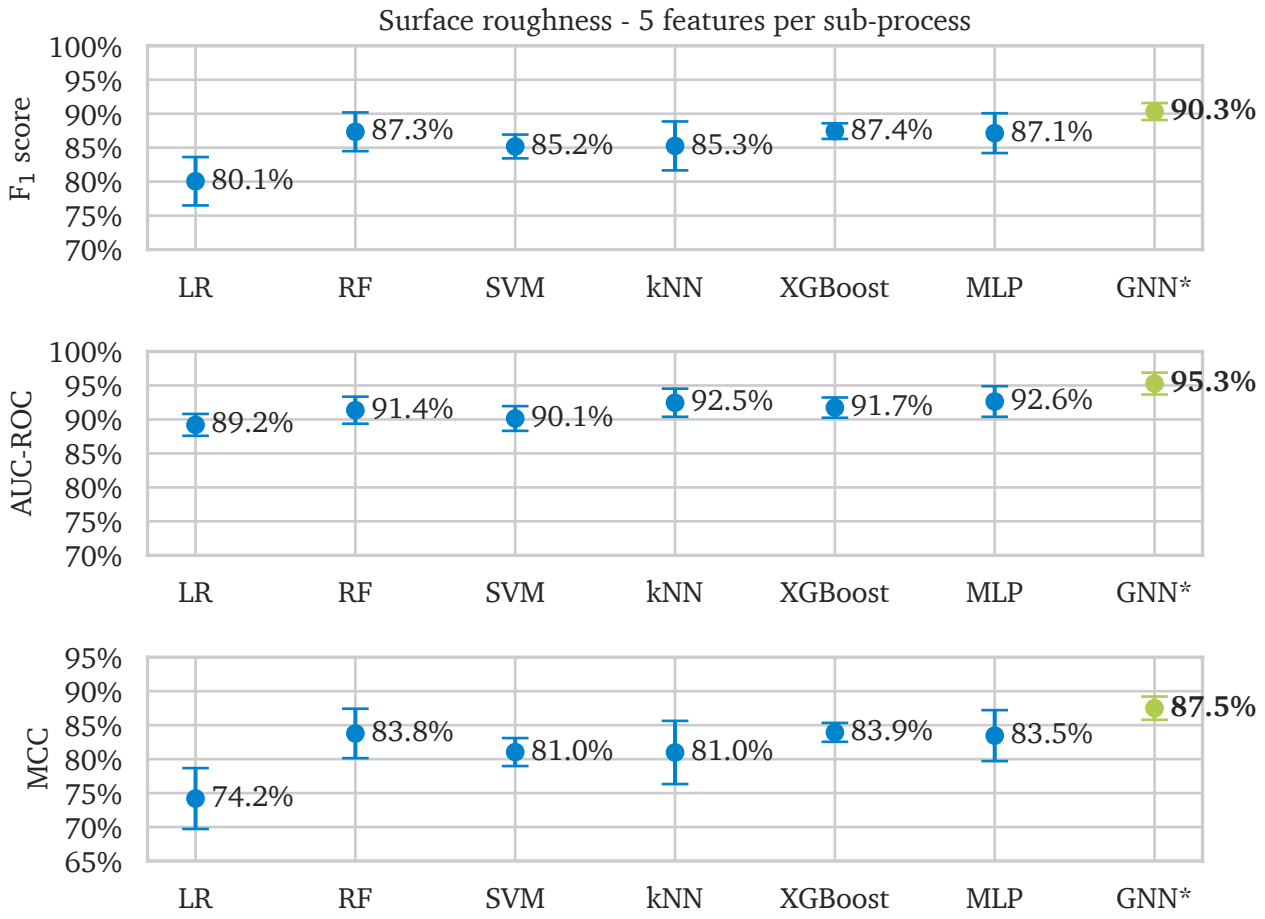


Figure C.3.: Performance comparison—Case study 2—Surface roughness (5 features). *Best performing model

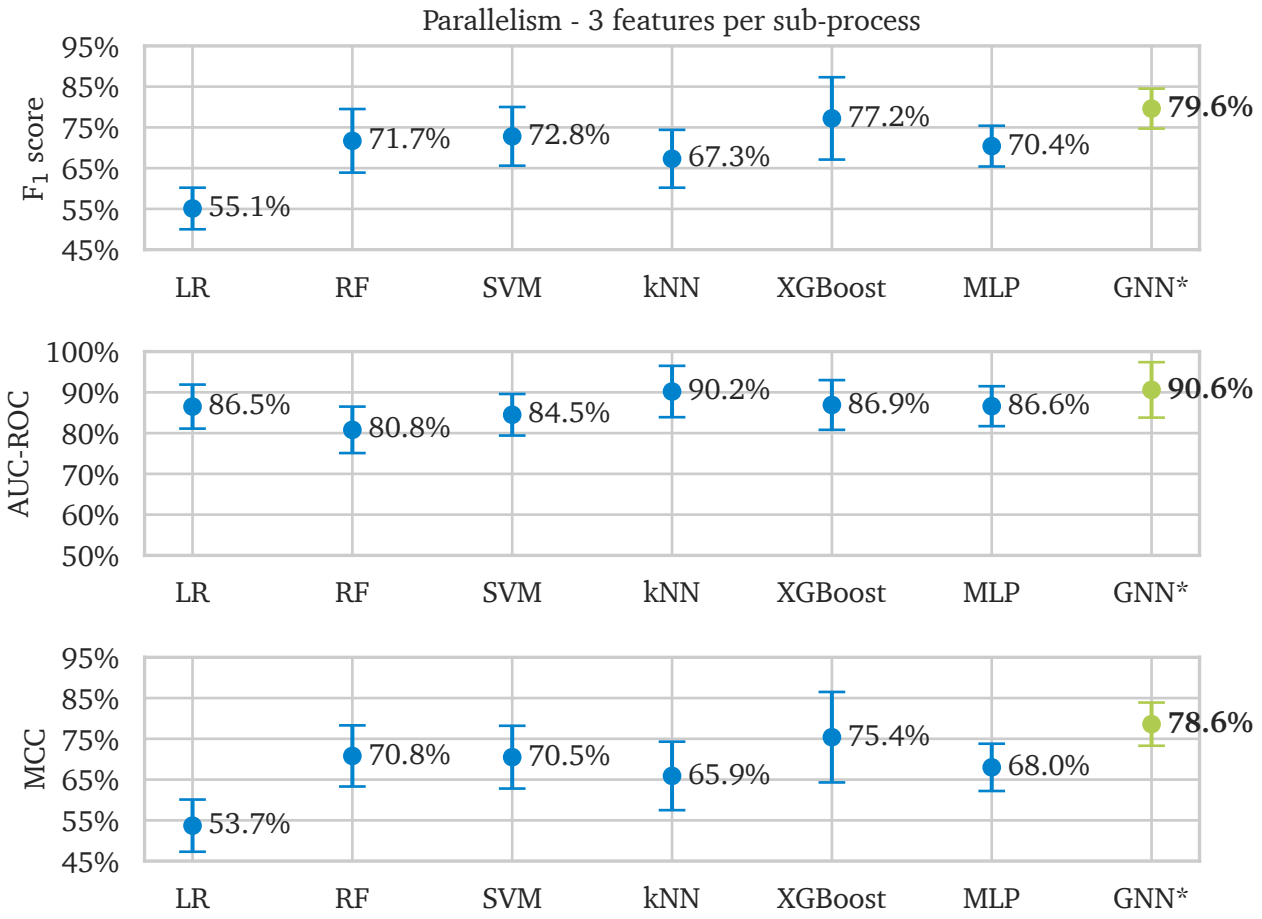


Figure C.4.: Performance comparison—Case study 2—Parallelism (3 features). *Best performing model

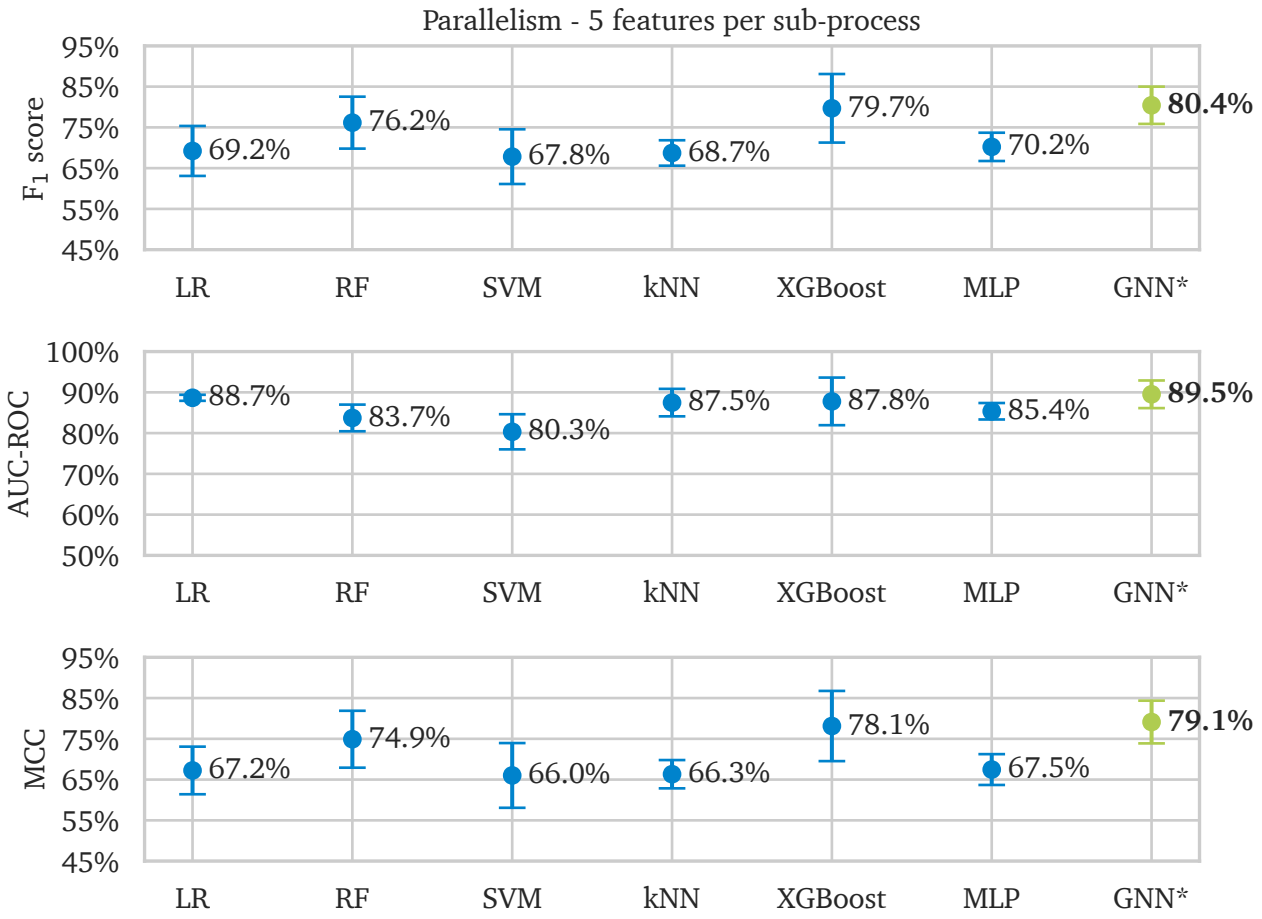


Figure C.5.: Performance comparison—Case study 2—Parallelism (5 features). *Best performing model

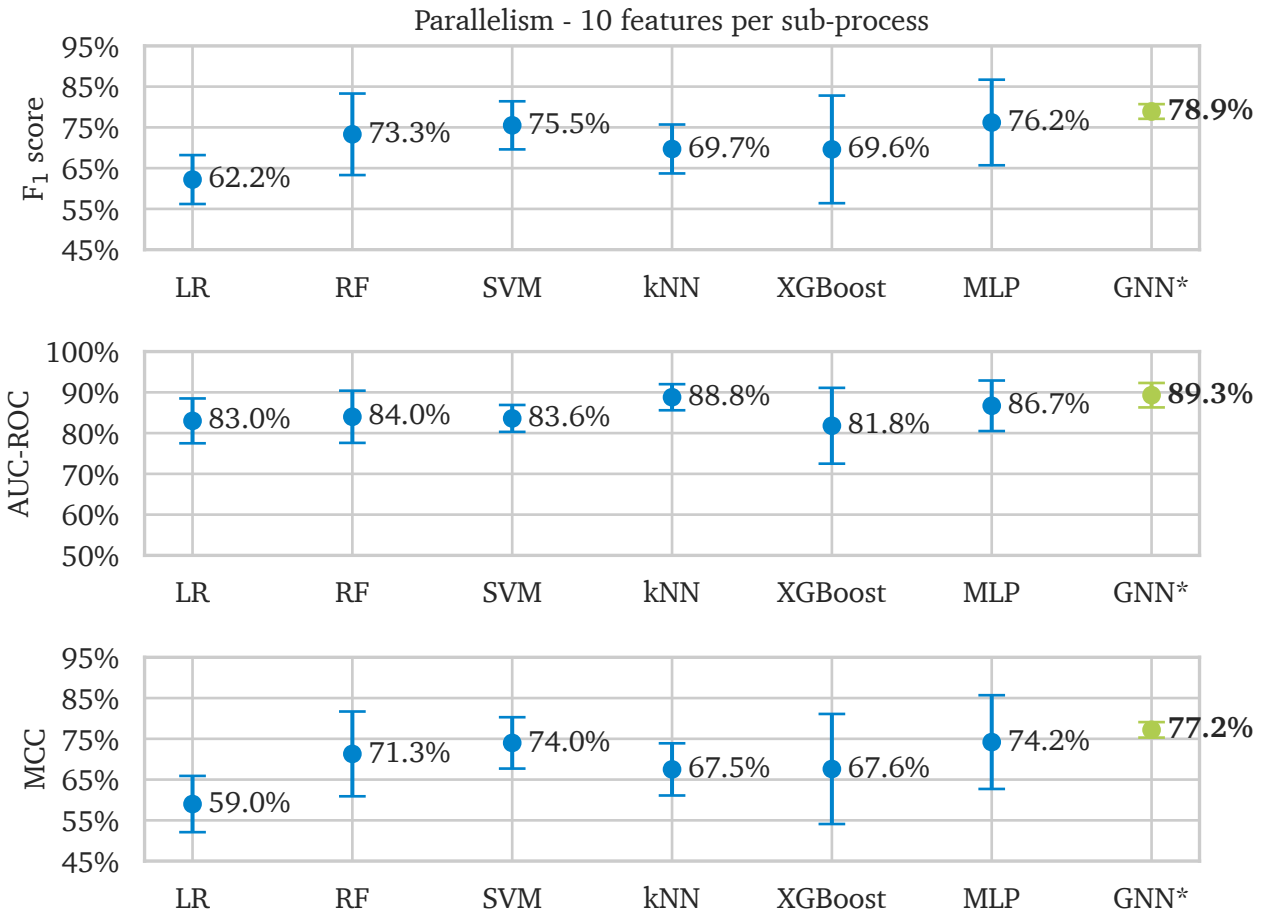


Figure C.6.: Performance comparison—Case study 2—Parallelism (10 features). *Best performing model

Bibliography

[1] Eberhard Abele, Joachim Metternich, and Michael Tisch. “Learning factories”. In: *Concepts, Guidelines, Best-Practice Examples* (2019).

[2] ACM Digital Library. *ACM Digital Library*. Visited on 07-07-2023. URL: https://dl.acm.org/.

[3] Fahmi Arif, Nanna Suryana, and Burairah Hussin. “A data mining approach for developing quality prediction model in multi-stage manufacturing”. In: *International Journal of Computer Applications* 69.22 (2013). ISSN: 0975-8887.

[4] Fahmi Arif, Nanna Suryana, and Burairah Hussin. “Cascade quality prediction method using multiple PCA+ ID3 for multi-stage manufacturing system”. In: *Ieri Procedia* 4 (2013), pp. 201–207. ISSN: 2212-6678.

[5] Mats Bagge, Mikael Hedlind, and Bengt Lindberg. “Tolerance chain design and analysis of in-process workpiece”. In: *Proceedings of the International Conference on Advanced Manufacturing Engineering and Technologies. Stockholm: KTH Royal Institute of Technology*. 2013, pp. 305–315.

[6] Geapa Batista and Diego Furtado Silva. “How k-nearest neighbor parameters affect its performance”. In: *Argentine symposium on artificial intelligence*. Citeseer, 2009, pp. 1–12.

[7] Peter W. Battaglia et al. “Relational inductive biases, deep learning, and graph networks”. In: *arXiv preprint arXiv:1806.01261* (2018).

[8] Ben Abdallah Ben Lamine, Sana, Malek Kamoua, and Haythem Grioui. “Quality prediction in a smart factory: a real case study”. In: *Proceedings of the 26th International Database Engineered Applications Symposium*. 2022, pp. 161–165. DOI: 10.1145/3548785.3548796.

[9] Yoshua Bengio, Aaron Courville, and Pascal Vincent. “Representation learning: A review and new perspectives”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828. ISSN: 0162-8828.

[10] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. ISBN: 0-387-31073-8.

[11] Alexander Bleakie and Dragan Djurdjanovic. “Feature extraction, condition monitoring, and fault modeling in semiconductor manufacturing systems”. In: *Computers in Industry* 64.3 (2013), pp. 203–213. ISSN: 01663615.

[12] Wayne C. Booth, Gregory Colomb, and Joseph M. Williams. *The Craft of Research*. Chicago USA: The University of Chicago Press, 2008.

[13] Bosch. *Bosch Production Line Performance: Reduce manufacturing failures*. Visited on 08-17-2023. URL: https://www.kaggle.com/c/bosch-production-line-performance/data.

[14] Leo Breiman. “Random forests”. In: *Machine learning* 45 (2001), pp. 5–32. ISSN: 0885-6125.

[15] Beatriz Bretones Cassoli, Nicolas Jourdan, and Joachim Metternich. “Knowledge Graphs for Data And Knowledge Management in Cyber-Physical Production Systems”. In: *Proceedings of the Conference on Production Systems and Logistics: CPSL 2022*. Hannover: publish-Ing, 2022, pp. 445–454.

[16] Beatriz Bretones Cassoli, Nicolas Jourdan, and Joachim Metternich. “Multi-source data modelling and graph neural networks for predictive quality”. In: *Procedia CIRP* 120 (2023), pp. 39–44. ISSN: 22128271.

[17] David Buterez et al. “Graph Neural Networks with Adaptive Readouts”. In: *arXiv preprint arXiv:2211.04952* (2022).

[18] Gürol Canbek et al. “Binary classification performance measures/metrics: A comprehensive visualized roadmap to gain new insights”. In: *2017 International Conference on Computer Science and Engineering (UBMK)*. IEEE, 2017, pp. 821–826.

[19] Girish Chandrashekar and Ferat Sahin. “A survey on feature selection methods”. In: *Computers & Electrical Engineering* 40.1 (2014), pp. 16–28. ISSN: 0045-7906.

[20] Chih-Chung Chang and Chih-Jen Lin. “LIBSVM: a library for support vector machines”. In: *ACM transactions on intelligent systems and technology (TIST)* 2.3 (2022), pp. 1–27. ISSN: 2157-6904.

[21] Bahzad Charbuty and Adnan Abdulazeez. “Classification based on decision tree algorithm for machine learning”. In: *Journal of Applied Science and Technology Trends* 2.01 (2021), pp. 20–28. ISSN: 2708-0757.

[22] Rongli Chen et al. “The Core Industry Manufacturing Process of Electronics Assembly Based on Smart Manufacturing”. In: *ACM Transactions on Management Information Systems* 13.4 (2023), pp. 1–19. ISSN: 2158-656X. DOI: 10.1145/3529098.

[23] Sejune Cheon et al. “Convolutional neural network for wafer surface defect classification and the detection of unknown defect class”. In: *IEEE Transactions on Semiconductor Manufacturing* 32.2 (2019), pp. 163–170. ISSN: 0894-6507. DOI: 10.1109/TSM.2019.2902657.

[24] Hao-Yi Chih et al. “Product quality prediction with convolutional encoder-decoder architecture and transfer learning”. In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2020, pp. 195–204. DOI: 10.1145/3340531.3412007.

[25] Maximilian Christ et al. “Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package)”. In: *Neurocomputing* 307 (2018), pp. 72–77. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2018.03.067.

[26] Marcello Colledani, Andrea Matta, and Tulio Tolio. “Analysis of the production variability in multi-stage manufacturing systems”. In: *CIRP annals* 59.1 (2010), pp. 449–452. ISSN: 0007-8506.

[27] Valter Costa et al. “Automatic Visual Inspection of Turbo Vanes produced by Investment Casting Process”. In: *Proceedings of the 2020 3rd International Conference on Sensors, Signal and Image Processing*. 2020, pp. 63–69. DOI: 10.1145/3441233.3441241.

[28] Verdiana Del Rosso et al. “Mechanical fault detection for induction motors based on vibration analysis: a case study”. In: *IECON 2021ÔÇô47th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2021, pp. 1–6. DOI: 10.1109/IECON48115.2021.9589189.

[29] Deutsches Institut für Normung e.V. *Qualitätsmanagementsysteme: Grundlagen und Begriffe*. Berlin, 2015.

[30] DGL. *GINConv*. Visited on 01-13-2023. URL: https://docs.dgl.ai/en/0.8.x/generated/dgl.nn.pytorch.conv.GINConv.html.

[31] DGL. *GraphConv*. Visited on 01-16-2023. URL: https://docs.dgl.ai/en/1.1.x/generated/dgl.nn.pytorch.conv.GraphConv.html.

[32] DGL. *SAGEConv*. Visited on 01-13-2023. URL: https://docs.dgl.ai/en/0.8.x/generated/dgl.nn.pytorch.conv.SAGEConv.html.

[33] Pradip Dhal and Chandrashekhar Azad. “A comprehensive survey on feature selection in the various fields of machine learning”. In: *Applied Intelligence* (2022), pp. 1–39. ISSN: 0924-669X.

[34] Jenny Diaz and Carlos Ocampo-Martinez. “Energy efficiency in discrete-manufacturing systems: Insights, trends, and control strategies”. In: *Journal of Manufacturing Systems* 52 (2019), pp. 131–145. ISSN: 02786125. DOI: 10.1016/j.jmsy.2019.05.002.

[35] Stefanos Doltsinis, Marios Krestenitis, and Zoe Doulgeri. “A machine learning framework for real-time identification of successful snap-fit assemblies”. In: *IEEE Transactions on Automation Science and Engineering* 17.1 (2019), pp. 513–523. ISSN: 1545-5955. DOI: 10.1109/TASE.2019.2932834.

[36] Guozhu Dong and Huan Liu. *Feature Engineering for Machine Learning and Data Analytics*. CRC press, 2018. ISBN: 1351721275.

[37] Stephan Dreiseitl and Lucila Ohno-Machado. “Logistic regression and artificial neural network classification models: a methodology review”. In: *Journal of biomedical informatics* 35.5-6 (2002), pp. 352–359. ISSN: 1532-0464.

[38] Shichang Du et al. “Markov modeling and analysis of multi-stage manufacturing systems with remote quality information feedback”. In: *Computers & Industrial Engineering* 88 (2015), pp. 13–25. ISSN: 0360-8352.

[39] Marc-André Filz, Jan Philipp Bosse, and Christoph Herrmann. “Digitalization platform for data-driven quality management in multi-stage manufacturing systems”. In: *Journal of Intelligent Manufacturing* (2023), pp. 1–20. ISSN: 1572-8145.

[40] Jerome H. Friedman. “Greedy function approximation: a gradient boosting machine”. In: *Annals of statistics* (2001), pp. 1189–1232. ISSN: 0090-5364.

[41] Yu-Kai Fu et al. “Event-triggered feedforward predictive control for dimension quality optimization in BIW assembly process”. In: *IEEE Transactions on Industrial Informatics* 18.2 (2022), pp. 1083–1090. ISSN: 1551-3203. DOI: 10.1109/TII.2021.3082187.

[42] Gianfranco Genta, Maurizio Galetto, and Fiorenzo Franceschini. “Inspection procedures in manufacturing processes: recent studies and research perspectives”. In: *International Journal of Production Research* 58.15 (2020), pp. 4767–4788. ISSN: 0020-7543.

[43] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O’Reilly Media, Inc., 2022. ISBN: 1098122461.

[44] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. ISBN: 0262337371.

[45] Denise Gosnell and Matthias Broecheler. *The Practitioner’s Guide to Graph Data: Applying Graph Thinking and Graph Technologies to Solve Complex Problems*. O’Reilly Media, Inc., 2020. ISBN: 1492044024.

[46] Will Hamilton, Zhitao Ying, and Jure Leskovec. “Inductive representation learning on large graphs”. In: *Advances in neural information processing systems* 30 (2017).

[47] William L. Hamilton. *Graph representation learning*. Vol. 14. Morgan & Claypool Publishers, 2020.

[48] Moneer Helu et al. “Industry Review of Distributed Production in Discrete Manufacturing”. In: *Journal of Manufacturing Science and Engineering* 142.11 (2020), p. 110802. ISSN: 1087-1357.

[49] Vitali Hirsch, Peter Reimann, and Bernhard Mitschang. “Data-driven fault diagnosis in end-of-line testing of complex products”. In: *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2019, pp. 492–503. DOI: 10.1109/DSAA.2019.00064.

[50] Vitali Hirsch, Peter Reimann, and Bernhard Mitschang. “Exploiting domain knowledge to address multi-class imbalance and a heterogeneous feature space in classification tasks for manufacturing data”. In: *Proceedings of the VLDB Endowment* 13.12 (2020), pp. 3258–3271. ISSN: 2150-8097. DOI: 10.14778/3415478.3415549.

[51] Vitali Hirsch et al. “Analytical approach to support fault diagnosis and quality control in End-Of-Line testing”. In: *Procedia CIRP* 72 (2018), pp. 1333–1338. ISSN: 22128271. DOI: 10.1016/j.procir.2018.03.024.

[52] IEEE Xplore. *IEEE Xplore*. Visited on 07-07-2023. URL: https://ieeexplore.ieee.org/.

[53] Malinka Ivanova and Nikolay Petkov. “Machine learning for in-circuit testing of printed circuit board assembly”. In: *Proceedings of the 2021 4th Artificial Intelligence and Cloud Computing Conference*. 2021, pp. 221–228. DOI: 10.1145/3508259.3508291.

[54] Jesus Barrasa, Amy E. Hodler, and and Jim Webber. *Knowledge Graphs: Data in Context for Responsive Businesses*. United States of America: O’Reilly Media, 2021. ISBN: 978-1-098-10483-2.

[55] Dan Jiang, Weihua Lin, and Nagarajan Raghavan. “A Gaussian mixture model clustering ensemble regressor for semiconductor manufacturing final test yield prediction”. In: *IEEE Access* 9 (2021), pp. 22253–22263. DOI: 10.1109/ACCESS.2021.3055433.

[56] Hideyuki Jippo et al. “Graph classification of molecules using force field atom and bond types”. In: *Molecular Informatics* 39.1-2 (2020), p. 1800155. ISSN: 1868-1743.

[57] Nicolas Jourdan et al. “A new benchmark dataset for machine learning applications in discrete manufacturing: CiP-DMD”. In: *Manuscript accepted for publication at the CIRP ICME Proceedings* (2023).

[58] Seokho Kang. “Joint modeling of classification and regression for improving faulty wafer detection in semiconductor manufacturing”. In: *Journal of Intelligent Manufacturing* 31.2 (2018), pp. 319–326. ISSN: 1572-8145. DOI: 10.1007/s10845-018-1447-2.

[59] Christoph Kellermann et al. “Fault Detection in Multi-stage Manufacturing to Improve Process Quality”. In: *2022 International Conference on Control, Automation and Diagnosis (ICCAD)*. IEEE, 2022, pp. 1–6. DOI: 10.1109/ICCAD55197.2022.9853909.

[60] Mohammed Al-Kharaz et al. “Data-Based Approach for Final Product Quality Inspection: Application to a Semiconductor Industry”. In: *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, 2021, pp. 1356–1362. DOI: 10.1109/CDC45484.2021.9683231.

[61] Kyoung-Yun Kim and Fahim Ahmed. “Semantic weldability prediction with RSW quality dataset and knowledge construction”. In: *Advanced Engineering Informatics* 38 (2018), pp. 41–53. ISSN: 14740346. DOI: 10.1016/j.aei.2018.05.006.

[62] Thomas N. Kipf and Max Welling. “Semi-supervised classification with graph convolutional networks”. In: *arXiv preprint arXiv:1609.02907* (2016).

[63] Thomas N. Kipf and Max Welling. “Semi-supervised classification with graph convolutional networks”. In: *ICLR 2017* (2017). URL: https://arxiv.org/abs/1609.02907.

[64] Barbara Kitchenham and Stuart Charters. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*. Ed. by In EBSE Technical Report, Software Engineering Group, School of Computer Science and Mathematics, Keele University, Department of Computer Science, University of Durham. 2007.

[65] Chakravanti Rajagopalachari Kothari. *Research methodology: Methods and techniques*. New Age International, 2004. ISBN: 8122415229.

[66] Oliver Kramer. “Scikit-learn”. In: *Machine learning for evolution strategies* (2016), pp. 45–53. ISSN: 33193338.

[67] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), pp. 436–444. ISSN: 0028-0836.

[68] Yazhou Li et al. “Surface quality evaluation based on roughness prediction model”. In: *Proceedings of the International Conference on Information Technology and Electrical Engineering 2018*. 2018, pp. 1–6. DOI: 10.1145/3148453.3306271.

[69] Yuanyuan Li et al. “Noncontact reflow oven thermal profile prediction based on artificial neural network”. In: *IEEE Transactions on Components, Packaging and Manufacturing Technology* 11.12 (2021), pp. 2229–2237. ISSN: 2156-3950. DOI: 10.1109/TCPMT.2021.3120310.

[70] Jinquan Lin et al. “Welding quality analysis and prediction based on deep learning”. In: *2021 4th World Conference on Mechanical Engineering and Intelligent Manufacturing (WCMEIM)*. IEEE, 2021, pp. 173–177. DOI: 10.1109/WCMEIM54377.2021.00045.

[71] Zhenyu Liu et al. “An adversarial bidirectional serial–parallel LSTM-based QTD framework for product quality prediction”. In: *Journal of Intelligent Manufacturing* 31.6 (2020), pp. 1511–1529. ISSN: 1572-8145.

[72] Marco Maggipinto et al. “A computer vision-inspired deep learning architecture for virtual metrology modeling with 2-dimensional data”. In: *IEEE Transactions on Semiconductor Manufacturing* 31.3 (2018), pp. 376–384. ISSN: 0894-6507. DOI: 10.1109/TSM.2018.2849206.

[73] Marco Maggipinto et al. “What are the most informative data for virtual metrology? A use case on multi-stage processes fault prediction”. In: *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2019, pp. 1796–1801. DOI: 10.1109/COASE.2019.8842942.

[74] Brian W. Matthews. “Comparison of the predicted and observed secondary structure of T4 phage lysozyme”. In: *Biochimica et Biophysica Acta (BBA)-Protein Structure* 405.2 (1975), pp. 442–451. ISSN: 0005-2795.

[75] L.E. Melkumova and S. Ya Shatskikh. “Comparing Ridge and LASSO estimators for data analysis”. In: *Procedia Engineering* 201 (2017), pp. 746–755. ISSN: 1877-7058.

[76] Richard Meyes et al. “A recurrent neural network architecture for failure prediction in deep drawing sensory time series data”. In: *Procedia Manufacturing* 34 (2019), pp. 789–797. ISSN: 2351-9789. DOI: 10.1016/j.promfg.2019.06.205.

[77] Rubén Moliner-Heredia et al. “A sequential inspection procedure for fault detection in multistage manufacturing processes”. In: *Sensors* 21.22 (2021), p. 7524. ISSN: 1424-8220. DOI: 10.3390/s21227524.

[78] Takuma Nagao et al. “Wafer-Level Characteristic Variation Modeling Considering Systematic Discontinuous Effects”. In: *Proceedings of the 28th Asia and South Pacific Design Automation Conference*. 2023, pp. 442–448. DOI: 10.1145/3566097.3567915.

[79] Takeshi Nakazawa and Deepak V. Kulkarni. “Wafer map defect pattern classification and image retrieval using convolutional neural network”. In: *IEEE Transactions on Semiconductor Manufacturing* 31.2 (2018), pp. 309–314. ISSN: 0894-6507. DOI: 10.1109/TSM.2018.2795466.

[80] Oliver Nalbach et al. “Predictive quality: Towards a new understanding of quality assurance using machine learning tools”. In: *W. Abramowicz & A. Paschke (Eds.), Business information systems* 320 (2018), pp. 30–42. URL: https://doi.org/10.1007/978-3-319-93931-5_3.

[81] William S. Noble. “What is a support vector machine?” In: *Nature biotechnology* 24.12 (2006), pp. 1565–1567. ISSN: 1087-0156.

[82] Moschos Papananias et al. “An intelligent metrology informatics system based on neural networks for multistage manufacturing processes”. In: *Procedia CIRP* 82 (2019), pp. 444–449. ISSN: 22128271. DOI: 10.1016/j.procir.2019.04.148.

[83] Chanhee Park et al. “Multitask learning for virtual metrology in semiconductor manufacturing systems”. In: *Computers & Industrial Engineering* 123 (2018), pp. 209–219. ISSN: 0360-8352. DOI: 10.1016/j.cie.2018.06.024.

[84] Iker Pastor-López et al. “How IoT and computer vision could improve the casting quality”. In: *Proceedings of the 9th International Conference on the Internet of Things*. 2019, pp. 1–8. DOI: 10.1145/3365871.3365878.

[85] Lakshmi N. Pedapudi et al. “Quality Prediction in Semiconductors using Unlabeled Data from Multiple Channels”. In: *2021 3rd International Conference on Management Science and Industrial Engineering*. 2021, pp. 200–206. DOI: 10.1145/3460824.3460855.

[86] Ricardo Silva Peres et al. “Multistage quality control using machine learning in the automotive industry”. In: *IEEE Access* 7 (2019), pp. 79908–79916. DOI: 10.1109/ACCESS.2019.2923405.

[87] Sebastian Raschka. “An overview of general performance metrics of binary classifier systems”. In: *arXiv preprint arXiv:1410.5330* (2014).

[88] Mohammad Rezaei-Malek et al. “A review on optimisation of part quality inspection planning in a multi-stage manufacturing system”. In: *International Journal of Production Research* 57.15-16 (2019), pp. 4880–4897. ISSN: 0020-7543.

[89] Hamidey Rostami, Jean-Yves Dantan, and Lazhar Homri. “Review of data mining applications for quality assessment in manufacturing industry: support vector machines”. In: *International Journal of Metrology and Quality Engineering* 6.4 (2015), p. 401. ISSN: 2107-6839.

[90] Ahmet Sahin et al. “A markovian approach for time series prediction for quality control”. In: *IFAC-PapersOnLine* 52.13 (2019), pp. 1902–1907. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2019.11.480.

[91] Benjamin Sanchez-Lengeling et al. “A gentle introduction to graph neural networks”. In: *Distill* 6.9 (2021), e33. ISSN: 2476-0757.

[92] Tobias Schlosser et al. “A novel visual fault detection and classification system for semiconductor manufacturing using stacked hybrid convolutional neural networks”. In: *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2019, pp. 1511–1514. DOI: 10.1109/ETFA.2019.8869311.

[93] Jacqueline Schmitt et al. “Predictive model-based quality inspection using Machine Learning and Edge Cloud Computing”. In: *Advanced Engineering Informatics* 45 (2020), p. 101101. ISSN: 14740346. DOI: 10.1016/j.aei.2020.101101.

[94] Robert Schmitt and Tilo Pfeifer. *Qualitätsmanagement: Strategien–Methoden–Techniken*. Carl Hanser Verlag GmbH Co KG, 2015. ISBN: 3446440828.

[95] ScienceDirect. *ScienceDirect*. Visited on 07-07-2023. URL: https://www.sciencedirect.com/.

[96] Scikit-learn. *Cross-validation: evaluating estimator performance*. Visited on 08-17-2023. URL: https://scikit-learn.org/stable/modules/cross_validation.html.

[97] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: from Theory to Algorithms*. Cambridge University Press, 2014. ISBN: 1139952749.

[98] Jianjun Shi. *Stream of variation modeling and analysis for multistage manufacturing processes*. CRC press, 2006. ISBN: 1420003909.

[99] Steven S. Skiena. *The Data Science Design Manual*. Springer, 2017. ISBN: 3319554441.

[100] Statista Research Department. *Daten & Fakten zum Fachkräftemangel in Deutschland*. Visited on 02-16-2024. URL: https://de.statista.com/themen/887/fachkraeftemangel/#topicOverview.

[101] Sebastian Stemmler et al. "Quality control in injection molding based on norm-optimal iterative learning cavity pressure control". In: *IFAC-PapersOnLine* 53.2 (2020), pp. 10380–10387. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2020.12.2777.

[102] Stefan Studer et al. "Towards CRISP-ML (Q): a machine learning process model with quality assurance methodology". In: *Machine learning and knowledge extraction* 3.2 (2021), pp. 392–413. ISSN: 2504-4990.

[103] Hasan Tercan and Tobias Meisen. "Machine learning and deep learning based predictive quality in manufacturing: a systematic review". In: *Journal of Intelligent Manufacturing* (2022), pp. 1–27. ISSN: 1572-8145. DOI: 10.1007/s10845-022-01963-8.

[104] Tze Chiang Tin et al. "The Implementation of a Smart Sampling Scheme C2O Utilizing Virtual Metrology in Semiconductor Manufacturing". In: *IEEE Access* 9 (2021), pp. 114255–114266. DOI: 10.1109/ACCESS.2021.3103235.

[105] tsfresh. *Overview on extracted features*. Visited on 01-16-2023. URL: https://tsfresh.readthedocs.io/en/latest/text/list_of_features.html.

[106] tsfresh. *tsfresh.feature_selection package*. Visited on 12-27-2023. URL: https://tsfresh.readthedocs.io/en/latest/api/tsfresh.feature_selection.html.

[107] Artem Turetskyy et al. "Battery production design using multi-output machine learning models". In: *Energy Storage Materials* 38 (2021), pp. 93–112. ISSN: 2405-8297. DOI: 10.1016/j.ensm.2021.03.002.

[108] Raphael Wagner et al. "Virtual in-line inspection for function verification in serial production by means of artificial intelligence". In: *Procedia CIRP* 92 (2020), pp. 63–68. ISSN: 22128271. DOI: 10.1016/j.procir.2020.03.126.

[109] Baicun Wang et al. "Smart manufacturing and intelligent manufacturing: A comparative review". In: *Engineering* 7.6 (2021), pp. 738–757. ISSN: 2095-8099.

[110] Minjie Yu Wang. "Deep graph library: Towards efficient and scalable deep learning on graphs". In: *ICLR workshop on representation learning on graphs and manifolds*. 2019.

[111] David H. Wolpert and William G. Macready. "No free lunch theorems for optimization". In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82. ISSN: 1089-778X.

[112] World Economic Forum in collaboration with Statista. *These are the top 10 manufacturing countries in the world*. Visited on 01-09-2023. URL: https://www.weforum.org/agenda/2020/02/countries-manufacturing-trade-exports-economics/.

[113] Lingfei Wu et al., eds. *Graph Neural Networks: Foundations, Frontiers, and Applications*. Singapore: Springer Nature Singapore, 2022. ISBN: 978-981-16-6054-2.

[114] Xiaofei Wu et al. "Development of convolutional neural network based Gaussian process regression to construct a novel probabilistic virtual metrology in multi-stage semiconductor processes". In: *Control Engineering Practice* 96 (2020), p. 104262. ISSN: 0967-0661. DOI: 10.1016/j.conengprac.2019.104262.

[115] Yichao Wu and Yufeng Liu. “Robust truncated hinge loss support vector machines”. In: *Journal of the American Statistical Association* 102.479 (2007), pp. 974–983. ISSN: 0162-1459.

[116] Yu Xiao and Maria Watson. “Guidance on conducting a systematic literature review”. In: *Journal of planning education and research* 39.1 (2019), pp. 93–112. ISSN: 0739-456X. DOI: 10.1177/0739456X1772397.

[117] Keyulu Xu et al. “How powerful are graph neural networks?” In: *arXiv preprint arXiv:1810.00826* (2018).

[118] Cheng-Han Yeh, Yao-Chung Fan, and Wen-Chih Peng. “Interpretable multi-task learning for product quality prediction with attention mechanism”. In: *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1910–1921. DOI: 10.1109/ICDE.2019.00207.

[119] Donghao Zhang et al. “Contrastive Decoder Generator for Few-shot Learning in Product Quality Prediction”. In: *IEEE Transactions on Industrial Informatics* (2022). ISSN: 1551-3203. DOI: 10.1109/TII.2022.3190554.

[120] Donghao Zhang et al. “Path enhanced bidirectional graph attention network for quality prediction in multistage manufacturing process”. In: *IEEE Transactions on Industrial Informatics* 18.2 (2022), pp. 1018–1027. ISSN: 1551-3203. DOI: 10.1109/TII.2021.3076803.

[121] H. C. Zhang and M. E. Huq. “Tolerancing techniques: the state-of-the-art”. In: *International Journal of Production Research* 30.9 (1992), pp. 2111–2135. ISSN: 0020-7543.

[122] Jianan Zhao et al. “Heterogeneous graph structure learning for graph neural networks”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 35. 2021, pp. 4697–4705.

[123] Li-Ping Zhao, Bo-Hao Li, and Yi-Yong Yao. “A novel predict-prevention quality control method of multi-stage manufacturing process towards zero defect manufacturing”. In: *Advances in Manufacturing* (2023), pp. 1–15. ISSN: 2095-3127. DOI: 10.1007/s40436-022-00427-9.

[124] Alice Zheng and Amanda Casari. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. O’Reilly Media, Inc., 2018. ISBN: 1491953195.

[125] Guoqiang Zhong et al. “An overview on data representation learning: From traditional feature learning to recent deep learning”. In: *The Journal of Finance and Data Science* 2.4 (2016), pp. 265–278. ISSN: 2405-9188.

[126] Jie Zhou et al. “Graph neural networks: A review of methods and applications”. In: *AI open* 1 (2020), pp. 57–81. ISSN: 2666-6510. DOI: 10.1016/j.aiopen.2021.01.001.

[127] Minghao Zhu et al. “The impact of intelligent manufacturing on labor productivity: An empirical analysis of Chinese listed manufacturing companies”. In: *International Journal of Production Economics* 267 (2024), p. 109070. ISSN: 0925-5273.