Institute of Software Technology

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Evaluation of MLOps Approaches and Implementation of a Data Product Development Pipeline

Leopold Ormos

Course of Study:	Informatik
Examiner:	Prof. Dr. Stefan Wagner
Supervisor:	Markus Haug, M.Sc, Dr. David Leiner
Commenced:	April 15, 2024
Completed:	October 15, 2024

Abstract

Recent advancements in Machine Learning (ML), particularly with Large Language Models (LLMs), have significantly impacted the tech industry, prompting companies to integrate these technologies into their products and services. However, many organizations lack the expertise to effectively develop or incorporate ML models, facing challenges in both development and operation. Despite the emergence of Machine Learning Operations (MLOps) paradigms intended to streamline the handling of Artificial Intelligence (AI)-enabled applications, selecting appropriate tools and constructing cohesive MLOps pipelines remain significant hurdles due to the vast and fragmented landscape of available tools.

This master's thesis investigates how a holistic MLOps pipeline can enhance the development and operational processes of data products, even in business contexts without extensive technical expertise. Conducted within a department of the Robert Bosch GmbH, the research is structured into three phases. The first phase involves a comprehensive evaluation of state-of-the-art MLOps architectures, platforms, and tools through a Rapid Review, establishing a foundation for the pipeline's design. The second phase focuses on designing and implementing an MLOps pipeline tailored to the specific requirements and constraints of the case study environment. The developed solution integrates with existing infrastructure on Azure and Development and Operations (DevOps) processes using GitHub and Terraform. It is built on the Databricks platform, utilizing integrated components such as Unity Catalog and MLflow, and is extended with the tool Langfuse and libraries like Giskard and Ragas. In the third phase, the implemented pipeline is evaluated in a practical setting through two hackathon-like use tests. Feedback from these tests assesses the pipeline's effectiveness in improving the efficiency and quality of data product development. The evaluation revealed successful application in data product development and identified areas for further improvement, such as enhancing the pipeline's documentation.

Overall, this thesis presents a complete process of developing an MLOps pipeline from initial evaluation to practical implementation and assessment in a real-world context. The findings contribute to the field by providing practical insights into effectively implementing MLOps pipelines, offering valuable guidance for organizations seeking to adopt MLOps practices to optimize their data product development.

Kurzfassung

Die jüngsten Fortschritte in Machine Learning (ML), insbesondere bezüglich Large Language Models (LLMs), haben die Tech-Industrie erheblich beeinflusst und Unternehmen dazu veranlasst, diese Technologien in ihre Produkte und Dienstleistungen zu integrieren. Viele Organisationen verfügen jedoch nicht über die Expertise, ML-Modelle effektiv zu entwickeln oder zu integrieren, und stehen vor Herausforderungen sowohl in der Entwicklung als auch beim Betrieb. Trotz des Aufkommens des Machine Learning Operations (MLOps)-Paradigma, das darauf abzielt, den Umgang mit Künstlicher Intelligenz (KI)-gestützen Anwendungen zu optimieren, bleiben die Auswahl geeigneter Tools und der Aufbau kohärenter MLOps-Pipelines bedeutende Hürden aufgrund der umfangreichen und fragmentierten Landschaft der verfügbaren Technologien.

Diese Masterarbeit untersucht, wie eine ganzheitliche MLOps-Pipeline die Entwicklungs- und Betriebsprozesse von Datenprodukten verbessern kann, selbst in Geschäftskontexten ohne umfangreiche technische Expertise. Die Forschung wurde in einer Abteilung der Robert Bosch GmbH durchgeführt und ist in drei Phasen strukturiert. Die erste Phase beinhaltet eine umfassende Bewertung aktueller MLOps-Architekturen, Plattformen und Tools durch einen Rapid Review, um eine Grundlage für das Design der Pipeline zu schaffen. Die zweite Phase konzentriert sich auf die Gestaltung und Implementierung einer MLOps-Pipeline, die auf die spezifischen Anforderungen und Einschränkungen der Fallstudienumgebung zugeschnitten ist. Die entwickelte Lösung integriert sich mit der bestehenden Infrastruktur auf Azure und den Development and Operations (DevOps)-Prozessen unter Verwendung von GitHub und Terraform. Sie basiert auf der Databricks-Plattform, nutzt integrierte Komponenten wie Unity Catalog und MLflow und wird durch das Tool Langfuse sowie Bibliotheken wie Giskard und Ragas erweitert. In der dritten Phase wird die implementierte Pipeline in einem praktischen Umfeld durch zwei hackathon-ähnliche Nutzertests evaluiert. Das aus diesen Tests gesammelte Feedback bewertet die Effektivität der Pipeline bei der Verbesserung der Effizienz und Qualität der Datenproduktentwicklung. Die Bewertung zeigt eine erfolgreich Anwendung in der Datenproduktentwicklung und identifizierte Bereiche für weitere Verbesserungen, wie beispielsweise eine verbesserte Dokumentation der Pipeline.

Insgesamt präsentiert diese Arbeit einen vollständigen Prozess der Entwicklung einer MLOps-Pipeline von der anfänglichen Auswertung bis zur praktischen Implementierung und Beurteilung in einem realen Kontext. Die Ergebnisse tragen zum Füllen einer Lücke in der Literatur bei, indem sie praktische Einblicke in die effektive Implementierung von MLOps-Pipelines bieten und wertvolle Erkenntnisse für Organisationen bereitstellen, die MLOps-Praktiken zur Optimierung ihrer Datenproduktentwicklung einführen möchten.

Acknowledgements

I would like to express my sincere gratitude to Prof. Dr. Wagner for examining this thesis despite his departure from the University of Stuttgart. His engaging lecture on *Software Engineering for AI-based Systems* inspired me to pursue the topic of MLOps for my master's thesis.

My heartfelt thanks go to my university supervisor, Markus Haug, for his extensive and excellent guidance throughout my thesis, especially his support regarding the use of scientific methods, and for his efforts in clarifying the examiner for this thesis.

I am also deeply grateful to my supervisor at Robert Bosch GmbH, Dr. David Leiner, for his exceptional mentorship and valuable advice during the execution and especially the writing of this thesis.

Furthermore, I would like to thank the VM/BDO department at Robert Bosch GmbH for providing the opportunity to conduct this thesis and supplying the infrastructure and resources necessary for its completion. I especially want to express my gratitude to the Datalab Team for warmly welcoming me into the team and for the always pleasant collaboration.

Additionally, I extend my sincere appreciation to all my colleagues at Bosch who provided initial input on the topic of MLOps at the start of this thesis, and especially to everyone who participated in the user tests.

Finally, a heartfelt thank you to my two supervisors, colleagues at Bosch, friends, and family for proofreading this thesis and offering helpful feedback for improvement.

Contents

1 Introduction **15**
- 1.1 Motivation . . . 15
- 1.2 Objective . . . 16
- 1.3 Case Study Environment . . . 17
- 1.4 Outline . . . 17

2 Background **19**
- 2.1 Data Product . . . 19
- 2.2 Machine Learning . . . 20
- 2.3 Development and Operations . . . 22
- 2.4 Machine Learning Operations . . . 24

3 Related Work **29**

4 Methodology **31**
- 4.1 Rapid Review . . . 32
- 4.2 Design Science . . . 36
- 4.3 Methods for Evaluation . . . 39

5 Evaluation of Machine Learning Operations approaches **45**
- 5.1 Preliminary . . . 45
- 5.2 Definition of Components . . . 49
- 5.3 Architecture Options . . . 52
- 5.4 Evaluation of Platforms . . . 55
- 5.5 Evaluation of Tools . . . 58

6 Design and Implementation of a Data Product Development Pipeline **67**
- 6.1 Design Process . . . 67
- 6.2 Final Architecture and Workflow . . . 77

7 Evaluation of the Data Product Development Pipeline **87**
- 7.1 Sample Use Cases . . . 87
- 7.2 Evaluation Results . . . 91
- 7.3 Analysis . . . 97

8 Discussion **103**
- 8.1 Evaluation of Machine Learning Operations approaches . . . 103
- 8.2 Design and Implementation of a Data Product Development Pipeline . . . 105
- 8.3 Evaluation of the Data Product Development Pipeline . . . 106
- 8.4 Threats to Validity . . . 107

9 Conclusion and Outlook **109**
9.1 Future Work 110

Bibliography **111**

A Requirements **117**

B Rapid Review Results **121**

C Architecture Decision Record Format **125**

D Identified Platforms and Tools **127**

E User Guide for the Template Repository **129**

F Sample Use Cases **131**
F.1 Use Case 1: Chatbot answering questions on a climate report 132
F.2 Use Case 2: Classification of handwritten digits 135

List of Figures

2.1 The hierarchy of Artificial Intelligence (AI) technologies: AI, Machine Learning (ML), Deep Learning, and Generative Artificial Intelligence (Gen AI) 20
2.2 Basic overview of a Retrieval Augmented Generation (RAG) setup: A retriever fetching knowledge related to the input question and a generator creating an answer from it. 21
2.3 The Development and Operations (DevOps) cycle illustrating the cyclical sequence of the DevOps phases *Plan*, *Code*, *Build*, *Test*, *Release*, *Deploy*, *Operate* and *Monitor*. 22
2.4 The role of Machine Learning Operations (MLOps) Engineer and its intersection with the adjacent roles Data Scientist, Data Engineer, Software Engineer, DevOps Engineer, and Backend Engineer. Adapted from [KKH23]. 24
2.5 A version of the MLOps cycle, building on the phases of the DevOps process and extended by a data and a model phase. Adapted from [SSV22]. 25
2.6 The technical MLOps components as defined by Kreuzberger et al. [KKH23]. Adapted from [KKH23]. 26
2.7 The MLOps stack as defined by Valohai [Sko20]. Adapted from [Sko20]. 27

4.1 Sequential order of the phases in this thesis 31
4.2 The phases of the design cycle as defined by Wieringa [Wie14]. 37

5.1 Distribution of different types of literature identified in the Rapid Review (white literature types depicted in light blue and grey literature types depicted in grey). . 48
5.2 Different ML model staging/deployment concepts [BKTT22]. 54

6.1 Architecture overview of MLOps infrastructure components together with infrastructure for a sample project. 78
6.2 Overview of MLOps pipeline with view on all stages. 81
6.3 Overview of MLOps pipeline with view on `DEV` stage. 82
6.4 Development flow for working with the template. 86

7.1 High-level architecture overview for sample use case 1. 88
7.2 High-level architecture overview for sample use case 2. 90
7.3 Tool-related problems per user group. 99
7.4 Process-related problems per user group. 100

C.1 The Architecture Decision Record (ADR) base template created for and used in this thesis. 125

F.1 MLOps workflow in Databricks for sample use case 1. 133
F.2 Frontend of the implemented sample use case 1: Chatbot answering questions on a climate report. 134

F.3 MLOps workflow in DATABRICKS for sample use case 2. 136
F.4 Frontend of the implemented sample use case 2: Classification of handwritten digits. 137

List of Tables

4.1 The inclusion criteria (IC) and exclusion criteria (EC) for the study selection in the Rapid Review. 33

5.1 Overview of defined MLOps components with priorities and if they are already covered by a tool. 51

6.1 Overview of MLOps components and selected platform or tools for each component as part of the MLOps pipeline. 71
6.2 List of requirements and their validation result 76

7.1 Roles and experience levels of test user group 1. 91
7.2 Roles and experience levels of test user group 2. 92
7.3 Amount of encountered issues for test group 1 categorized by type of problem and separated by observation and involvement. 93
7.4 Amount of encountered issues for test group 2 categorized by type of problem and separated by observation and involvement. 93
7.5 Results of the anonymous survey. 96

A.1 List of requirements 117

B.1 White literature selected in Rapid Review 121
B.2 Grey literature selected in Rapid Review 122

Acronyms

ACM Association for Computing Machinery. 34

ADR Architecture Decision Record. 9

AI Artificial Intelligence. 3, 9

API Application Programming Interface. 15

CD Continuous Delivery & Deployment. 22

CI Continuous Integration. 22

CT Continuous Training. 25

DevOps Development and Operations. 3, 9

Gen AI Generative Artificial Intelligence. 9

IaC Infrastructure as Code. 58

IEEE Institute of Electrical and Electronic Engineers. 34

IP Internet Protocol. 78

LLM Large Language Model. 3, 15

LLMOps Large Language Model Operations. 19

ML Machine Learning. 3, 9

MLOps Machine Learning Operations. 3, 9

MLR Multivocal Literature Review. 29

PAT Personal Access Token. 80

RAG Retrieval Augmented Generation. 9

SLR Systematic Literature Review. 29

UI User Interface. 61

1 Introduction

1.1 Motivation

Recent advancements in ML, particularly the progress with Large Language Models (LLMs), have revolutionized the tech industry. Companies are increasingly investing in these technologies to enhance their products and services. Incorporating ML and LLMs into product development has reached a point where it can significantly improve various aspects of traditional applications. To remain competitive, businesses must focus on leveraging these technologies effectively.

However, many companies often lack the necessary expertise not only to develop ML models professionally but even to integrate and utilize pre-existing open-source models or models accessed via Application Programming Interfaces (APIs) effectively in their products and services. This can be see as only 36% of ML projects are deployed beyond pilot stage according to McKinsey [PSSW21] and a study by Gartner says that only 53% of AI projects make it from prototype to production [Gar20].

This challenge extends beyond development to the sustainable maintenance of these models. Despite the advent of the MLOps paradigm, designed to streamline the development and deployment of AI-enabled applications, numerous challenges persist [LCB20] [BJS19]. A particularly daunting challenge is selecting appropriate tools and their combination into a cohesive MLOps process. The MLOps landscape is vast and fragmented, with many tools available for every stage of the ML development lifecycle. This overwhelming variety makes it nearly impossible for companies to maintain a comprehensive overview of all available tools, let alone choose the most suitable ones for their respective needs. The diversity and complexity of these tools, coupled with the rapid evolution of the field, complicate the task of integrating them into a coherent, end-to-end MLOps pipeline that aligns with business objectives and operational requirements.

While there are existing proposals for MLOps setups and complete platforms, these solutions are often tailored to the specific requirements of individual projects, which vary widely in scope, objectives, and constraints. Thus, it is crucial to define each scenario's constraints and thoroughly evaluate potential solutions and tools to effectively address these unique needs. Although numerous papers provide theoretical discussions on MLOps, identify challenges, or offer comparisons of various tools, there is a lack of practical implementations and, more importantly, evaluations of the effectiveness of designed MLOps pipelines in real-world contexts, making it difficult to draw meaningful conclusions about their practical value. Therefore, it is essential to conduct practical implementations and thorough evaluations in actual business settings to better understand the real-world impact and effectiveness of MLOps pipeline solutions for developing data products.

1.2 Objective

Given the diverse and often project-specific nature of MLOps solutions, it is crucial to evaluate their efficiency in real-world settings to understand their practical value. Therefore, the objective of this master's thesis is to investigate how a holistic MLOps pipeline can improve the development process of data products, even in business contexts without in-depth technical expertise. The aim is to provide an MLOps solution that streamlines ML operational processes, ensuring reproducibility, scalability, efficient model deployment, testing strategies, and monitoring. This leads to the following overarching research question:

How can an MLOps pipeline be leveraged to improve the development and operational processes of data products?

To answer this question, this thesis is structured into three main phases, with distinct goals and specific research questions:

Goal 1: The goal of the first phase is to establish a comprehensive understanding of the current state-of-the-art MLOps approaches, which is necessary for the later design of an MLOps pipeline. This involves a thorough evaluation of the landscape of MLOps architectures, platforms, and tools. The research questions (RQ) for this part are:

RQ1 What are the architectural options and relevant components for building an MLOps pipeline?

RQ2 Which platforms or tools are suitable for the different components of an MLOps pipeline?

Goal 2: Building on the insights obtained regarding relevant MLOps architectures and components and suitable tools, the second phase addresses the design problem of developing a data product development pipeline. The goal is to create an MLOps pipeline suitable in the context of the case study environment described in Section 1.3. This results in the following design problem (DP):

DP1 Design and implement a data product development pipeline tailored to the specific requirements and constraints of the target environment.

Goal 3: The goal of the final phase is to evaluate the effectiveness of the implemented MLOps pipeline in a practical setting. This involves collecting feedback from a representative target user group which, test the solution to determine how well the MLOps pipeline improves the development process of data products in terms of efficiency and quality. This is described through the following research question (RQ):

RQ3 How effective is the implemented MLOps pipeline in improving the development process of data products in a practical setting?

By addressing these research questions and design problems, this thesis aims to provide a complete analysis of MLOps pipelines, from initial tool and architecture evaluation to real-world implementation and a final assessment of effectiveness. This approach will help determine such pipelines' practical value and impact in optimizing data product development.

1.3 Case Study Environment

To apply the development and evaluation of such an MLOps pipeline to a practical setting, this master's thesis is conducted within a department of the Robert Bosch GmbH. This specific case study environment and its requirements influence all the research goals of this thesis, from evaluating suitable MLOps tools to designing a fitting data product development pipeline to the assessment of the MLOps pipeline in a practical setting.

The department focuses on digitalization and undertakes complete projects utilizing cutting-edge technologies. It also provides comprehensive guides, reference architectures, and blueprints to facilitate the adoption of these technologies across other departments. They serve as models for digital transformation initiatives and play a crucial role in disseminating knowledge and best practices throughout the organization. For them, developing an efficient MLOps pipeline is thus essential not only to improve and streamline their processes but also to create a blueprint and guidelines that can be utilized across other departments.

In conclusion, designing and evaluating an MLOps pipeline is crucial for improving the department's workflow and efficiency. Additionally, creating a reusable template for other departments can significantly benefit the organization, fostering standardization and best practices with the development and deployment of data products.

1.4 Outline

This thesis first provides a brief overview of the fundamentals necessary for further understanding this work as shown in Chapter 2, Background. Following that, Chapter 3 discusses related work in the field of MLOps. This ranges from general papers providing an introduction to the topic of MLOps over papers that define challenges and best practices to other work that focuses on the implementation and evaluation of concrete MLOps pipelines. Chapter 4 explains the scientific approach and methods and the reasons why they are chosen are explained in detail. This includes a Rapid Review, a Design Science approach, and evaluation methods.

Then, Chapter 5 describes the evaluation of different MLOps approaches. This includes an initial collection of requirements set as criteria for a comprehensive assessment of architecture components and various MLOps platforms and tools. Afterward, the design and implementation of the data product development pipeline is described in Chapter 6. Based on the insights achieved in the previous chapter, a solution for the design of an MLOps pipeline is developed. Additionally, the implementation of this MLOps pipeline is covered in this chapter with a focus on the lessons learned during implementation and the extent to which these influenced the final data product development pipeline. Chapter 7 then describes evaluating the implemented data product development pipeline and assessing the results to determine to what extent the designed solution simplifies and accelerates the development process of data products and whether it improves the quality and maturity of the developed data products.

Finally, Chapter 8 discusses the achieved results and possible threats to the validity of this work, and Chapter 9 provides a conclusion and outlook on future work on this topic.

2 Background

This chapter introduces and provides an overview of the fundamental concepts that will be relevant for the remainder of this thesis. It begins by defining the term *data product* in the scope of this work in Section 2.1. Following this, the field of ML is briefly described in Section 2.2, highlighting the different types of ML. Section 2.3 then introduces the concept of DevOps, explaining its principles and advantages in the software development lifecycle. Building on these concepts, Section 2.4 concludes with an overview of the field of MLOps and one of its subfields Large Language Model Operations (LLMOps). This section outlines the different phases and components of MLOps and explains its significance in managing ML workflows effectively.

2.1 Data Product

A data product is often seen as a dataset asset, although this is only a subset of a data product and is more precisely described by the term *data as a product* [Rig21]. Generally, a data product is a digital tool, system, or service designed to deliver value for a specific end goal by leveraging data [Pat12]. These products are built around the processes of data collection, processing, analysis, and presentation, all aimed at addressing specific user needs. In essence, a data product is a software application or system that transforms raw data, often in real-time, into valuable, actionable insights, services, or capabilities for its users [Tea24]. The development of data products encompasses the entire lifecycle, from identifying use cases and collecting data to deploying and maintaining the product.

In the context of this thesis, the term *data product* refers specifically to applications or products that incorporate some form of ML component. A data product can, for example, be based on a classical ML model trained on an existing dataset, or it can involve using an LLM. This allows the creation of tools such as intelligent chatbots. Therefore, in this thesis, the term *data product* encompasses not only web applications but also chatbots and similar intelligent systems that use data together with ML to provide enhanced functionality and value beyond what traditional applications offer.

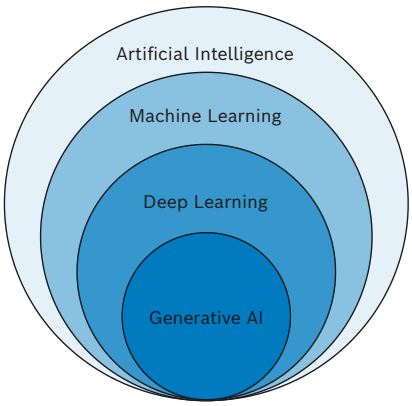


Figure 2.1: The hierarchy of AI technologies: AI, ML, Deep Learning, and Gen AI

2.2 Machine Learning

The overarching concept of AI describes the use of technology to enable computers to imitate the cognitive abilities of human intelligence, for example, the capacity to understand and respond to written or spoken language [Goo]. As visualized in Figure 2.1, ML is a subset of AI that enables computer systems to improve their performance on specific tasks through experience [Goo]. Instead of being explicitly programmed, ML systems learn from data, identifying patterns and making decisions with minimal human intervention [Goo]. Within the domain of ML, Deep Learning represents a more specialized subset that uses neural networks to model complex patterns. Nested inside is Gen AI, a specialized form of Deep Learning that generates new content, such as text or images, by learning from vast datasets. The field of Gen AI, together with LLMs and the concept of RAG, is further described in Section 2.2.1.

The typical ML process includes model requirements, data collection, data cleaning, data labeling, feature engineering, model training, model evaluation, model deployment, and model monitoring [ABB+19]

ML can be categorized into three main types based on the nature of the data used [Mur22]:

- **Supervised Learning**: This type uses labeled data, where each example in the dataset is paired with a ground truth label. The algorithm learns to map an input to the correct output. Typical applications include classification and regression tasks.
- **Unsupervised Learning**: This type deals with unlabeled data. The algorithm tries to identify patterns and relationships within the data without predefined labels. Techniques such as clustering and association are often used, with applications in customer segmentation and market basket analysis.
- **Reinforcement Learning**: In this type, an agent learns to make decisions by interacting with an environment and receives feedback through rewards or penalties. It is widely used in robotics or game playing.

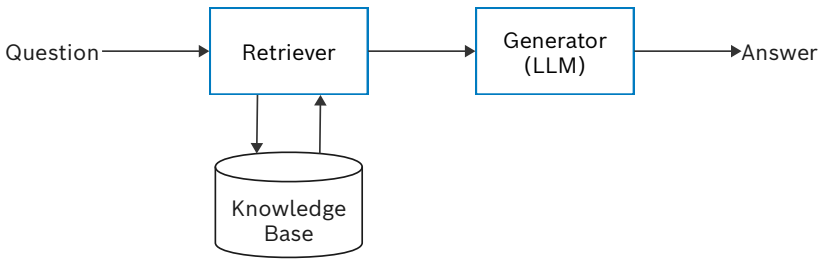


Figure 2.2: Basic overview of a RAG setup: A retriever fetching knowledge related to the input question and a generator creating an answer from it.

2.2.1 Generative Artificial Intelligence, Large Language Models and Retrieval-augmented Generation

Recent advances in the field of ML have given rise to Gen AI. Gen AI encompasses technologies that can create new content, such as text, images, or audio, based on patterns learned from large datasets. As a prominent aspect of Gen AI, significant attention has been brought to LLMs [ZZL+23]. LLMs are designed to understand and generate human language at scale. Models are trained on vast datasets containing diverse text sources and can perform a variety of natural language processing tasks, including translation, summarization, and text generation [ZZL+23]. These models leverage deep learning architectures, particularly transformer models, to capture the complexities of language [ZZL+23].

One notable concept in Gen AI is RAG systems. RAG systems combine the strengths of retrieval-based components and generation-based models [GXG+24]. As depicted in Figure 2.2, an input question is first inserted into a retriever component, which identifies relevant information from a knowledge base [GXG+24]. As preparation for the retrieval process, the documents of the knowledge base are chunked (divided into smaller parts), transformed into embeddings (high-dimensional numerical vectors) capturing their semantic meaning, and stored in a vector database (specialized in storing high-dimensional data). During the retrieval process, the question is converted into an embedding and similarity algorithms are used to search for matching embeddings in the vector database, returning relevant results. Then, the initial question and the retrieved knowledge are inserted into a generator component using an LLM to generate an accurate and contextually relevant answer [GXG+24]. This approach is particularly effective in scenarios requiring up-to-date or specific information, such as question answering or customer support [GXG+24].

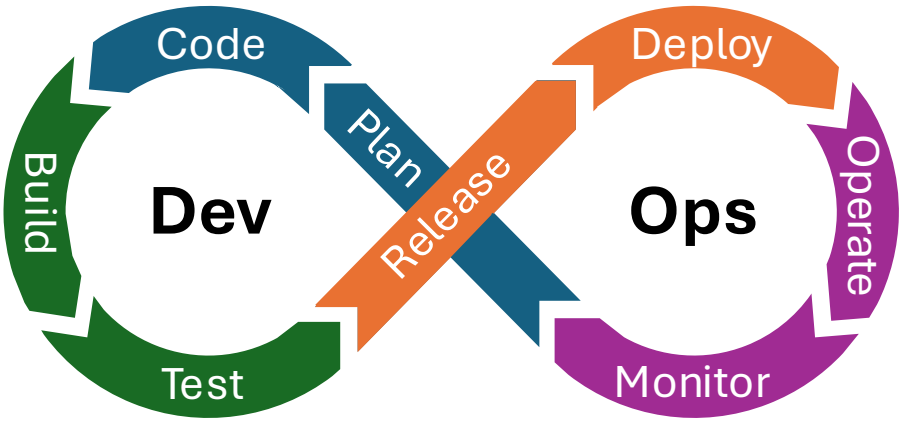


Figure 2.3: The DevOps cycle illustrating the cyclical sequence of the DevOps phases *Plan*, *Code*, *Build*, *Test*, *Release*, *Deploy*, *Operate* and *Monitor*.

2.3 Development and Operations

The term DevOps is a combination of *Development* and *Operations*, representing a paradigm shift in software development that combines these previously separate domains. DevOps emerged around 2008/2009 as a concept aimed at reducing inefficiencies like slow development cycles or miscommunication by bridging the gap between development and operations teams [KKH23] [Mez].

Traditionally, software development and operations were distinct phases, with separate teams responsible for each. Development teams would create applications and hand them over to operations teams during pre-planned, infrequent releases. The operations team would then manage the infrastructure and maintain the application in production. This separation often led to communication challenges, knowledge transfer issues, and slow release cycles.

DevOps addresses these challenges by promoting a culture of collaboration, communication, and knowledge sharing between development and operations [KKH23]. It emphasizes the integration of development, quality assurance, and operations into a single continuous process [ASS+24]. This integration enables faster and more efficient delivery cycles, increased team productivity, and improved overall software quality [EAN+17] [FS17].

A key aspect of DevOps is the emphasis on automation and continuous processes, particularly through Continuous Integration (CI) and Continuous Delivery & Deployment (CD) practices [KKH23]. CI involves frequent code integration and merging, leading to faster and more efficient delivery cycles [FS17] [Bos14]. CD ensures an automated release processes, enabling fast, frequent, and reliable releases [KKH23].

The DevOps approach typically follows a cyclical process, often referred to as the DevOps cycle as depicted in Figure 2.3. While there are slight variations in how the phases of this cycle are named, a common version encompasses the following phases, as described by Moreschi et al. and Subramanya et al. [MRL+23] [SSV22]:

- **Plan**: All activities preceding code development, such as requirements analysis, with the goal of creating a product roadmap and defining project scope.
- **Code**: Everything related to code development, including version control, code reviews, and collaboration among developers.
- **Build**: The process of creating a build artifact from code stored in a repository, often involving compilation and packaging of the application.
- **Test**: The phase where the application is tested, typically in a staging environment, for functionality, security, and performance to ensure quality and reliability.
- **Release**: After successful build and testing, this phase involves planning and preparing for the application's production release, including versioning and documentation.
- **Deploy**: The often automated and tool-supported deployment process where the application is made available in the production environment.
- **Operate**: In this phase, the application is in use by customers and is managed and maintained by the operations team, ensuring availability and performance.
- **Monitor**: The continuous process of observing and analyzing the entire pipeline based on data collected during operation, enabling feedback for continuous improvement.

The cyclical nature of the DevOps process emphasizes that development does not end with the deployment of an application. Instead, it continues iteratively, with insights from operations feeding back into planning and development. In organizations where the release process is highly automated, DevOps practices can lead to continuous delivery of software products [AA21]. Additionally, DevOps contributes to enhanced collaboration and improved operations [EAN+17]. By adopting these practices, organizations aim to reduce the time-to-market [FS17] [EAN+17] [ZBC16], while increasing the quality of software releases and reducing the risk of failures and downtime [BWZ15].

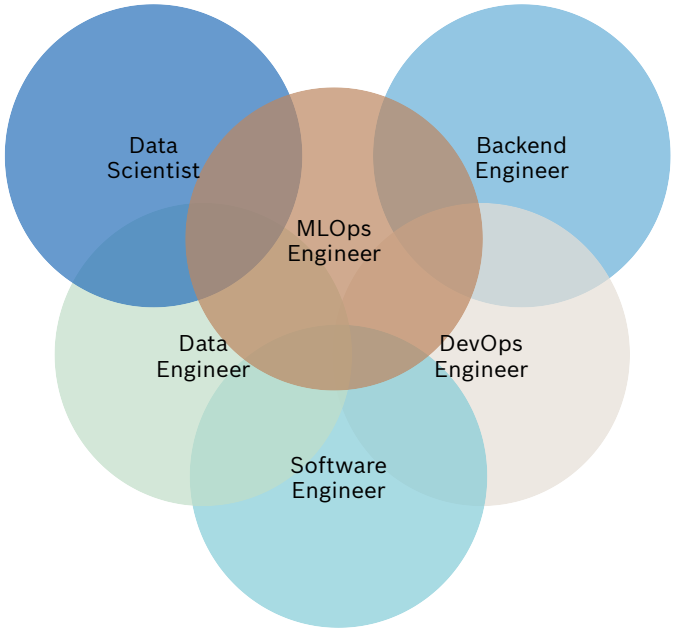


Figure 2.4: The role of MLOps Engineer and its intersection with the adjacent roles Data Scientist, Data Engineer, Software Engineer, DevOps Engineer, and Backend Engineer. Adapted from [KKH23].

2.4 Machine Learning Operations

While effective for software development, traditional DevOps practices are not sufficient for ML development because they do not address the unique challenges of ML workflows, such as handling large datasets, continuous model training, versioning of models, and monitoring model performance over time. These challenges require specialized approaches that go beyond classical DevOps. And precisely for this, the term MLOps, a combination of *Machine Learning* and *Operations*, describes a paradigm that applies DevOps principles to ML and the development of AI-enabled applications [ZYD20]. It aims to automate ML processes, development, and deployment using DevOps practices and approaches [SNKP22]. MLOps can be viewed as an intersection between ML and DevOps [AA21]. Definitions vary, however, as Kreuzberger et al. and Testi et al. describe it as the combination of ML, DevOps, and data engineering [KKH23] [TBF+22]. Regardless of the precise definition, the MLOps concept aims at speeding up an ML model's lengthy operationalizing by integrating tailored DevOps practices [RMRO21], allowing teams to concentrate on leveraging the latest technologies and optimizing model performance [AA21].

When looking at the role of the MLOps Engineer, it is evident that MLOps is a combination of different fields. Kreuzberger et al. describe this role as a link between the roles of Data Scientist, Data Engineer, Software Engineer, DevOps Engineer, and Backend Engineer, as illustrated in Figure 2.4, highlighting the complexity of both the role of MLOps Engineer and the MLOps field itself [KKH23].

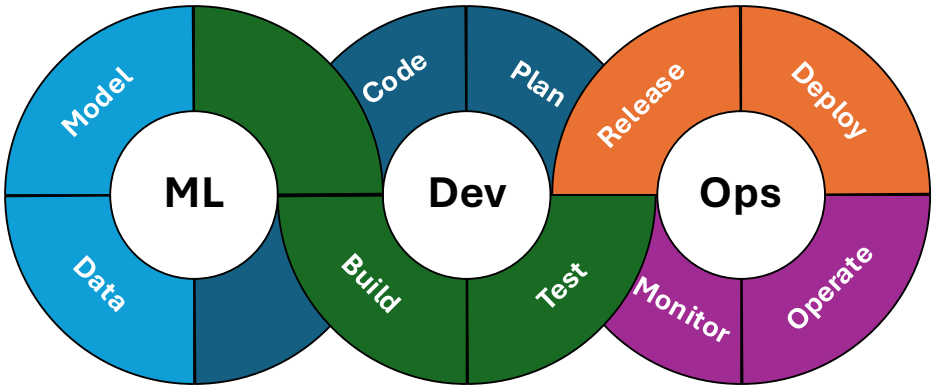


Figure 2.5: A version of the MLOps cycle, building on the phases of the DevOps process and extended by a data and a model phase. Adapted from [SSV22].

2.4.1 Machine Learning Operations Phases

Similar to the phases of the DevOps process described in Section 2.3, different definitions exist for the phases of the MLOps cycle. One common approach is to adapt and extend the established phases of DevOps [SSV22]. This means adding *data* and *model* phases for ML applications and additional tasks to the existing *test*, *deploy*, and *monitor* phases [SNKP22] [SSV22].

- **Data** (new): All data related steps, from an explorative data analysis, to extraction, validation and preparation of datasets.
- **Model** (new): This phase includes the model development, training, and evaluation on test datasets.
- **Test** (extended): The testing phase is extended by the validation and testing of ML models.
- **Deploy** (extended): In addition to deploying the application, an automated and tool-supported process for deploying and serving ML models is needed.
- **Monitor** (extended): The monitoring phase additionally continuously checks the ML model regarding performance and data drift.

As a result, the eight-step DevOps process becomes a ten-step MLOps process. In addition to the familiar concepts of CI and CD, Continuous Training (CT) becomes relevant in the MLOps process, describing the aim of continuously and automatically retraining models [SNKP22] [RMRO21].

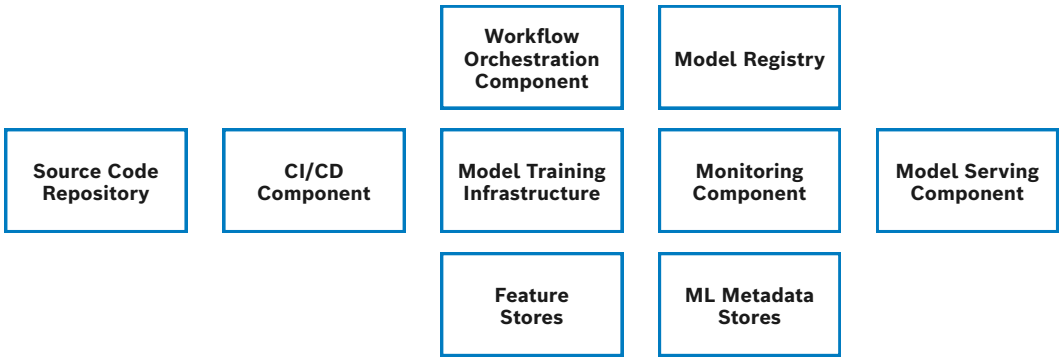


Figure 2.6: The technical MLOps components as defined by Kreuzberger et al. [KKH23]. Adapted from [KKH23].

2.4.2 Machine Learning Operations Components

Instead of viewing the MLOps workflow from a phase perspective, it can also be represented based on components. Likewise, although the underlying concept is always similar, there is no uniform definition of MLOps components in the literature. Figure 2.6 shows a breakdown of the MLOps components as defined by Kreuzberger et al. [KKH23], and Figure 2.7 depicts the MLOps stack as defined by Valohai [Sko20]. As can be seen, many identical or similar components appear in both versions, but some components only occur in one version. Both overviews include components already known from the DevOps process, such as a source code repository or a monitoring component, as well as new components like a model registry or a serving component [KKH23].

All these MLOps components will be explored in greater detail in the main body of this thesis, where a standard definition of MLOps components is defined in Section 5.2 and respective possible tools for the use in an MLOps pipeline are examined in Section 5.5.

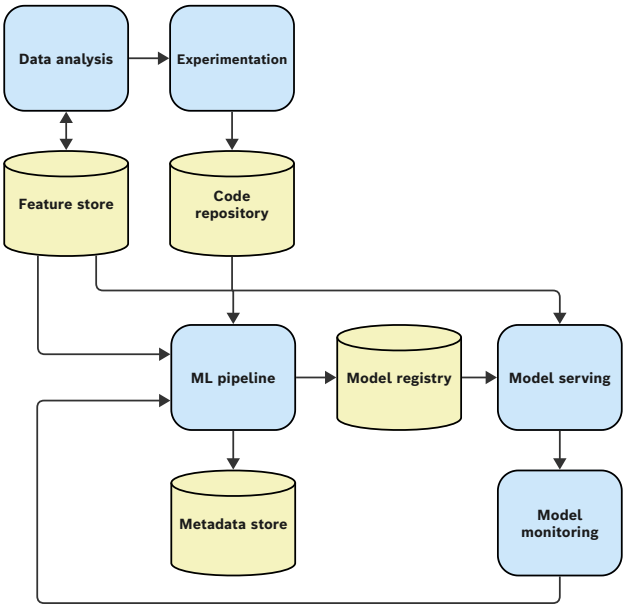


Figure 2.7: The MLOps stack as defined by Valohai [Sko20]. Adapted from [Sko20].

2.4.3 Large Language Model Operations

As described in Section 2.2.1, there has been a rise in the field of Gen AI and LLMs lately. Compared to more straightforward, smaller ML models, the larger and more complex LLMs lead to additional challenges in their training and usage. For this reason, LLMOps emerges as a subfield of MLOps, focusing on the development and operationalization of LLMs. While some components of classical MLOps pipelines remain relevant or gain prominence in LLMOps, others become less significant, especially when using LLMs in RAG systems.

For instance, evaluating and monitoring results from an LLM-based system is crucial for detecting hallucinations in LLM responses, requiring different capabilities than evaluating traditional ML projects. Additionally, organizations must navigate the complex landscape of data privacy, bias mitigation, and transparency while leveraging the transformative potential of these language models [KSKG23]. In addition, more data and a more powerful training infrastructure are required to train LLMs compared to simple ML models. On the other hand, the component of ML model development may not be required at all if only pre-trained foundation models, possibly directly via an API, are used.

This shift in focus necessitates adaptations to existing MLOps practices to accommodate the unique challenges and requirements of LLMs, for example, in RAG systems.

3 Related Work

Starting around 2019 and 2020, the term MLOps began appearing increasingly in the literature. Consequently, numerous papers now define the concept of MLOps, provide an overview of the field, and explain its importance.

For example, Alla and Adari explain the concepts of MLOps, its usefulness, and provide insights on how it can be implemented with varying degrees of automation [AA21]. They argue that MLOps is crucial because it significantly eases the deployment and maintenance of ML solutions by automating most parts, enabling teams to keep up with the latest technology and produce best-performing results [AA21]. Similarly, Kreuzberger et al. offer another definition of MLOps, along with an overview of its principles, components, roles, and architecture through a literature review, tool review, and expert interviews. Their guidelines on automating and operating ML products are technology-agnostic, and they discuss the selection of open-source tools versus enterprise solutions or a combination of both [KKH23]. In addition to providing a definition of MLOps and its components, Symeonidis et al. focus on the challenges and available tools and how to select them. They note that many advanced open-source tools in the field often focus on specific tasks within an MLOps pipeline rather than covering the end-to-end lifecycle of ML applications [SNKP22].

Two early papers, one by Lwakatare et al. from 2020 and one by Baier et al. from 2019, discuss the challenges during the development and operation of AI-enabled applications [LCB20] [BJS19]. Lwakatare et al. propose solutions to these challenges driven by the idea of combining DevOps and ML workflow processes [LCB20]. On the other hand, while focusing on trends and challenges, Tamburri specifically addresses the challenges of developing and maintaining a sustainable MLOps process [Tam20].

Beyond identifying challenges, several papers offer solutions such as best practices, implementation guidelines, or reference architectures for implementing MLOps systems. For instance, Matsui and Goya present five steps to guide the understanding and practice of MLOps and assist with its effective implementation [MG22]. John et al. conducted a Systematic Literature Review (SLR) and grey literature review to derive a framework for adopting MLOps depending on the current state of a company. Additionally, they present a maturity model outlining the different stages companies can be in with their MLOps adoption: automated data collection, automated model deployment, semi-automated model monitoring, and fully automated model monitoring [JOB21].

Focusing more on tools, Ruf et al. provide a concrete approach for selecting the right open-source MLOps tools. They emphasize that due to the multitude of available tools and the growing complexity of MLOps systems, it is essential to invest sufficient time in planning the MLOps process and selecting the tools or platforms [RMRO21]. Further papers dealing with tools in the MLOps field and their comparison are by Recupito et al. and by Moreschi et al., with both employing the method of a Multivocal Literature Review (MLR), which is a form of an SLR including grey literature [GFM19]. Recupito et al. look at 13 different MLOps tools and their functionalities in detail [RPC+22]. One of their results is that most tools support similar features but apply different

approaches, providing individual advantages depending on the concrete requirements [RPC+22]. Moreschi et al. identify 84 MLOps tools, map them to their purpose, and investigate possible incompatibilities [MRL+23]. An interesting finding is that the majority of end-to-end platforms do not commonly integrate with other tools, while no incompatibilities among other tools were found allowing the combination of multiple tools from different phases into a seamless MLOps pipeline [MRL+23].

Although not many papers focus on evaluating MLOps solutions, Zhou et al. conducted a case study implementing a functional ML platform with Kubeflow and compared time and performance implications in detail [ZYD20].

With the recent advent of LLMs, the field of MLOps has expanded to include the operationalization of LLM-based systems. Due to its different focus and requirements, this specific field is covered by the term LLMOps. Kulkarni et al. explore this area, providing a reference framework for the stack of LLM-based applications and addressing newly important components and aspects such as vector databases and testing LLM-generated responses [KSKG23].

In conclusion, a substantial body of foundational work defines and explains the concept of MLOps, along with several tool comparisons and studies that outline best practices and guidelines for implementing MLOps systems. In addition, some papers include practical components detailing the implementation of either part of an MLOps pipeline or a complete MLOps pipeline. These studies often focus on specific scenarios with external dependencies and requirements for which an appropriate MLOps solution is designed, consistent with the approach taken in this thesis. However, what is less often explored is an evaluation of the designed solution, especially how the implemented data product development pipeline improves the developer's workflow or leads to shorter development time and improved results. This gap highlights the need for more comprehensive studies that not only propose MLOps architectures but also implement, test, and evaluate them in real-world settings, drawing practical conclusions and learnings from such evaluations.

This thesis aims to fill this gap by not only designing and implementing a holistic MLOps pipeline but also evaluating and testing it on the development of example use cases within the context of the Robert Bosch GmbH. This approach will provide valuable insights into the effectiveness of the MLOps solution, offering a practical reference for both the academic community and industry practitioners.

4 Methodology

This chapter outlines the different methodologies used to accomplish the objectives of this thesis. As described in Section 1.2 and depicted in Figure 4.1, the research is structured into three sequential phases, each addressing specific research questions or design problems.

To address the first research goal of evaluating state-of-the-art MLOps approaches and answering the research questions **RQ1** and **RQ2**, a Rapid Review [CPS20] is performed as described in Section 4.1. This is used to gather a sufficient decision-making foundation for the subsequent MLOps pipeline design. It is combined with an initial collection of requirements using semi-structured interviews. To document the different options and their respective degree of fulfillment of these requirements, ADRs are applied.

Building on the findings from the Rapid Review, the next phase consists of the design problem **DP1** of designing and implementing a data product development pipeline. This phase is guided by the principles of Design Science [Wie14], as shown in Section 4.2, and contains the iterative development and adaptions to achieve the final MLOps pipeline. The resulting tool and architecture decisions are documented in the respective ADRs.

The third phase addresses the subsequent goal of assessing the effectiveness of the MLOps pipeline in improving the development of data products in practice. To answer this research question **RQ3**, feedback and insights are collected from a setup where a user group tests the MLOps pipeline by implementing an example use case, as outlined in Section 4.3.

Applying all these methods thus makes it possible to answer how holistic MLOps pipelines can be designed, implemented, and evaluated to optimize the development process of data products.



Figure 4.1: Sequential order of the phases in this thesis

4.1 Rapid Review

To achieve the first research goal of this thesis, the evaluation of different MLOps approaches, a Rapid Review is performed. Initially applied in the medical field, a Rapid Review is a lightweight secondary study methodology that can be conducted with reduced time investment and is characterized by methodological restrictions [CPS20] [Mat17]. Cartaxo et al. described the application of the Rapid Review methodology to the Software Engineering domain, defining its goal as supporting decision-making for specific practical problems with a clear objective in a timely manner [CPS20]. Due to time and effort constraints, Rapid Reviews often involve limitations in the research and acquisition of literature [CPS20] [Mat17]. However, a Rapid Review generally follows the structure of a Systematic Review, particularly in defining search terms and inclusion or exclusion criteria.

A contentious point in Rapid Reviews is the inclusion of grey literature. Cartaxo et al. advise against using grey literature, arguing that the Rapid Review methodology already compromises scientific rigor to some extent, and including grey literature would further amplify this [CPS20]. On the other hand, Garousi et al. describe the benefits and considerations of using grey literature, stating that researchers can increasingly benefit from its inclusion [GFMR20]. Based on an initial exploratory search for potential sources in both white and grey literature on the topic, it becomes apparent that while several papers exist on these subjects, as described in Chapter 3, there is also promising grey literature available. Further, other papers conducting literature reviews on this topic, such as Recupito et al. also incorporate grey literature to evaluate current MLOps tooling.

Therefore, the inclusion of grey literature in this Rapid Review is justified for several reasons:

- MLOps solution providers, especially large cloud providers and many smaller open-source solutions, offer comprehensive documentation, tool comparisons, reference architectures, and application examples. This wealth of information is based on their practical experiences with MLOps.
- The field of MLOps, while already offering many solutions and tools, is still evolving at a rapid pace. Some current tools or approaches are often not yet, or only minimally, represented and addressed in existing white literature.

By including grey literature in the Rapid Review, not only the knowledge from the practical experiences of major cloud providers on MLOps but also information on the latest developments and tools are getting incorporated.

This knowledge base obtained through the Rapid Review answers the research goal of evaluating different MLOps approaches and, more precisely, the research questions defined in Section 1.2. The first research question, **RQ1**, seeks to research architectural options and determine the different components in an MLOps setup. The following research question, **RQ2**, aims to provide an overview and comparison of different MLOps platforms and tools for each MLOps component identified in **RQ1**.

ID	Criteria	Applied to	Justification
Inclusion Criteria (IC)			
IC1	The study contains information about MLOps architectures or components	white & grey	Has to contain relevant information regarding at least one of the research questions.
IC2	The study contains information about MLOps framework or tool evaluations or comparisons	white & grey	Has to contain relevant information regarding at least one of the research questions.
Exclusion Criteria (EC)			
EC1	Results not in English language	white & grey	To ensure accessibility and correct comprehension.
EC2	Results from earlier than 2020	white & grey	The term MLOps became prevalent from 2020 onward; earlier works often use different definitions focusing only on ML model operation.
EC3	No citations (only applied if citation information available at search engine)	white	Necessary to maintain scope and only include the most influential studies; uncited works may be less established. New, not yet cited, work is also excluded, but latest information is still included via grey literature.
EC4	Non Computer Science domain (only applied if criteria available at search engine)	white	To focus on studies within the Computer Science field relevant to MLOps.
EC5	Sponsored results	grey	To avoid advertisements or promotional content that could bias the review.
EC6	Entries from outside of the top 10 search results	grey	To limit results and maintain a manageable scope for the Rapid Review.

Table 4.1: The inclusion criteria (IC) and exclusion criteria (EC) for the study selection in the Rapid Review.

Based on the procedures proposed by Kitchenham for systematic reviews [Kit04], the following three steps are executed for the Rapid Review:

1. Defining the search strings
2. Applying the search terms on the chosen search engines
3. Filtering out and selecting the relevant results based on inclusion and exclusion criteria

Although these steps and the defined criteria are similar for both white and grey literature, there are some differences in their application: For the white literature review, the search terms used are *MLOps* and *LLMOps*. They are applied as a search in the title to the search engines of the Association for Computing Machinery (ACM), Institute of Electrical and Electronic Engineers (IEEE), and Springer. The search is constrained to these search engines as one way to restrict this Rapid Review's scope and time effort. For the review of grey literature, a private Google tab is used as the search engine. The applied search terms are:

- MLOps tools comparison
- LLMOps tools comparison
- MLOps architecture guide
- filetype:pdf MLOps whitepaper architecture

This modification of search terms for the grey literature search is necessary due to the fundamental differences in search engines. Searching for just MLOps or LLMOps on Google leads to significantly more results, many of which are too general and not sufficiently targeted to the research questions.

The inclusion criteria (IC) and exclusion criteria (EC) shown in Table 4.1 are used to filter and select the final list of studies. The exclusion criteria are applied directly during the search process, with at least one criterion matching leading to an exclusion. Adherence to the inclusion criteria is ensured by a subsequent screening of the results, with at least one of the criteria needing to match.

Executing the Rapid Review with this configuration and inclusion and exclusion criteria makes it possible to identify resources highly relevant to the research goal of evaluating different MLOps approaches while simultaneously limiting the scope and effort to a suitable level.

4.1.1 Requirement Analysis

At the outset of this work, the exact requirements for the MLOps pipeline to be developed are not clearly defined. This mirrors the expected situation in other real-world setups. Therefore, the requirement analysis is an explicit part of this work, as it follows real-world practices and ensures the development of a solution that meets the specific goals and constraints of the case study environment.

To gather the different requirements, semi-structured interviews were conducted with multiple team members of the case study department to collect various requirements. In consultation with the responsible parties from the case study department, a final list of requirements is defined. This includes a categorization by type of requirement as well as assigning a priority to each requirement.

The priority levels of the requirements are divided into three levels: *High*, *Medium*, and *Low*. As there are different types of requirements, their consolidation includes a classification into several categories based on their nature and implications for the MLOps pipeline design:

- **Defined Constraints**: These are technical dependencies derived from the existing infrastructure and organizational policies. They ensure the developed solution integrates seamlessly into the existing environment with minimal adaptions. While these constraints may limit the solution space, they enable a detailed evaluation of the remaining options within the scope of this work. For these reasons, this category is called *Defined Constraints*.
- **Functional Requirements**: The directly measurable requirements are categorized as *Functional Requirements*. These must be met for the solution to be viable and can be directly applied during the design of the MLOps pipeline.
- **Non-functional Requirements**: The requirements that are not as easily measurable, especially during the design of the MLOps pipeline before implementation and testing, are categorized as *Non-functional Requirements*. While non-functional requirements should typically still be defined in a way that makes it possible to measure their fulfillment, it is expected that this may not be possible here during the initial requirement analysis. Instead, these provide a collection of qualities relevant to the developed MLOps pipeline. In combination with the priorities, this can be used to define which qualities are most important.

4.1.2 Architecture Decision Records

To ensure a structured evaluation and transparency of all architecture and tool comparisons as well as later decisions, ADRs are applied. This method is used in the industry, for example, as part of the arc42[1] documentation template, to make informed decisions and document them. Further, ADRs are already in use within the case study environment. An ADR can vary in complexity and can thus be used for both minor and major decisions. The fundamental structure of an ADR involves describing a problem within a specific context that needs to be solved, for which different options are available. The different options, along with their advantages and disadvantages, are listed and can be compared based on defined criteria, such as, for example, effort, time, or cost. Finally, not only is the decided option documented, but the rationale behind the choice, along with any dependencies or potential impacts on other decisions, is also included. In a corporate context, where the environment or external regulations may change, an ADR provides a solid foundation to revisit the problem domain, potentially consider additional options, or choose a different option based on changed requirements or new knowledge.

[1] https://docs.arc42.org/tips/9-5/

4.2 Design Science

After an extensive evaluation of different MLOps approaches and tool comparisons, the second research objective can be tackled. As described in Section 1.2, the second research goal is the design problem, **DP1**, of designing and implementing a data product development pipeline matching the requirements of the case study environment. To achieve this research objective, the methodology of Design Science research and its design cycle are employed. Design Science describes the design and investigation of artifacts in a given problem context with the aim of improving something within that context [Wie14]. This approach is particularly suitable for addressing complex, real-world problems by creating and evaluating of innovative artifacts.

Design Science has two types of research problems: design problems and knowledge questions [Wie14]. Design problems aim for a change in the real world and require an analysis of stakeholder goals. A solution to a design problem can be any kind of artifact, including even a conceptual design [Wie14]. One characteristic of design problems is that there are usually multiple possible solutions [Wie14].

The goal of this section, to design and implement a data product pipeline, aligns well with such a design problem. The solution aims to improve the development and operation of AI-enabled applications within a specific context. Furthermore, the final form of the solution is dependent on the goals of the department, and there could be many potential designs.

Wieringa provides a template for defining design problems [Wie14]:

- improve <a problem context>
- by <(re)designing an artifact>
- that satisfies <some requirements>
- in order to <help stakeholders achieve some goals>

Applying this template to the design problem of this work results in the following description:

- improve the development of data products
- by designing and implementing an MLOps pipeline
- that satisfies the requirements of the development team
- in order to enable faster and higher quality development of data product use cases

An essential characteristic of Design Science is that it is not limited to addressing a single type of problem; rather, one design problem might depend on previous knowledge questions and frequently lead to other design problems or knowledge questions [Wie14]. For instance, in this study, the execution of the Rapid Review to evaluate different MLOps approaches and the research questions **RQ1** and **RQ2** can be viewed as preceding knowledge questions that inform the subsequent design problem of creating an MLOps pipeline. Similarly, the third research goal, which involves evaluating the effectiveness of the designed MLOps pipeline (as addressed by research question **RQ3**), can be seen as a knowledge question that emerges from this design problem and the necessity for evaluating the MLOps pipeline. This transition from a design problem to a related knowledge question is a

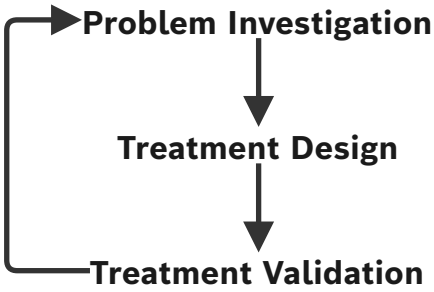


Figure 4.2: The phases of the design cycle as defined by Wieringa [Wie14].

typical pattern in Design Science Research [Wie14]. In the further course of this work, however, the individual research objectives are treated separately, with the Design Science methodology only applied to the design problem of the second research goal.

Design Science Research is inherently iterative, often involving a cycle of designing artifacts and investigating their effectiveness in a given context [Wie14]. Wieringa describes this with the engineering cycle consisting of the following parts [Wie14]:

- *Problem investigation*: What needs to be improved and why?
- *Treatment design*: Design artifacts that could treat the problem
- *Treatment validation*: Would these designs treat the problem?
- *Treatment implementation*: Apply the designed artifact to the problem
- *Implementation evaluation*: How successful is the applied treatment?

Wieringa uses the term *treatment* to describe that, although solutions are being designed, the focus is on the interaction of the artifact with a problem context [Wie14]. For design problems, only the first three tasks of the engineering cycle apply: *problem investigation*, *treatment design*, and *treatment validation* [Wie14]. These three tasks seen in Figure 4.2 form what Wieringa describes as the design cycle [Wie14]. This design cycle can be performed multiple times or in parallel with numerous artifact proposals to refine the solution. Once a validated treatment is achieved at the end of the design process, it can become part of the broader engineering cycle, which includes *Treatment Implementation* and *Implementation Evaluation* in a real user context. While the Design Science methodology is only applied to the design problem in the second research goal of this thesis (Goal 2), the third research goal of evaluating the implemented MLOps pipeline (Goal 3) can thus be seen as fulfilling the final two phases of the engineering cycle.

The following describes how the three tasks of the design cycle are applied to the design problem of developing a data product development pipeline.

Problem Investigation

The first task of the design cycle is to understand the problem thoroughly. This involves identifying the stakeholders and defining their goals and requirements. In this context, it is also crucial to consider the constraints, technical limitations, or opportunities presented by the case study

environment. Especially important is identifying the target end user group, as this directly influences the prioritization of requirements and, consequently, the design of the MLOps pipeline. By clearly defining the problem and understanding the needs and constraints, a more effective solution can be developed.

Treatment Design

The second task of the design cycle, *treatment design*, begins with defining the requirements for the MLOps pipeline based on stakeholder goals and the constraints of the case study environment. This phase primarily involves developing new MLOps pipeline concepts that effectively meet these requirements. It includes combining different MLOps platforms and tools and integrating them into a coherent MLOps workflow. The selection of these platforms and tools is informed by the insights of the previous research objective of evaluating different MLOps approaches. Given the complexity of MLOps pipelines, multiple design options may emerge, often with slight variations, such as selecting one tool over another for a specific functionality. Additionally, this phase considers existing MLOps tools or processes already in use within the case study environment, like CI/CD processes, which could be integrated into or reused in the design of the data product development pipeline.

Treatment Validation

The third task of the design cycle, *treatment validation*, focuses on validating the different versions of MLOps pipelines developed in the *treatment design* step. This involves assessing the feasibility of the proposed options based on the defined requirements. Given the complexity of deploying and integrating various platforms and tools in an MLOps pipeline, theoretical validation alone is insufficient. Instead, the infrastructure components of the MLOps pipeline, such as platforms and deployed tools, are validated by implementing a functional setup. During this practical implementation phase, challenges and issues are expected to arise, potentially influencing the final design of the MLOps pipeline. To manage these challenges, structured field notes are maintained, documenting each issue or challenge with details, including the topic, a description of the issue, the actions taken and possible solutions considered, the final decision, and the impact on the design. While not all problems will impact the final design, this documentation is crucial for justifying decisions influenced by practical implementation. Additionally, the pipeline's process and workflow are validated to ensure that all necessary MLOps components and phases are included and that the defined constraints and requirements are met. This validation confirms that the final MLOps pipeline is technically sound, implementable in practice, and aligned with stakeholder goals and the limitations of the case study environment.

In conclusion, by applying this Design Science Research approach, this section seeks to contribute to the practical improvement of data product development processes through the creation of an MLOps pipeline artifact. As the next step, the third research goal provides an evaluation of the effectiveness of the proposed MLOps pipeline in a practical setting.

4.3 Methods for Evaluation

Once the MLOps pipeline is designed and implemented, the next step is to evaluate the developed data product development pipeline to achieve the third research goal. While a theoretical validation addressing the initial requirements is already covered by the treatment validation task of the design step, the MLOps pipeline's effectiveness when applied by users in a practical setup remains open. To answer the research question **RQ3** defined in Section 1.2, the MLOps pipeline is evaluated with test users using the data product development pipeline for sample use cases.

Section 4.3.1 describes the prerequisites and preparations for applying the MLOps pipeline in a user test. Section 4.3.2 provides an in-depth look at the evaluation methods employed. It includes participant observation while users interact with the MLOps pipeline and a modified focus group interview to gather final feedback and direct opinions from participants. Finally, Section 4.3.3 explains how the analysis and interpretation of the received feedback is performed.

4.3.1 Evaluation Setup

The overall setup for performing the evaluation of the MLOps pipeline consists of a group of developers testing the developed data product development pipeline for implementing a sample use case in a hackathon-like setup. This is done to prevent the MLOps pipeline from just being looked at from a theoretical point of view and not being tested in practice. As the scope of the evaluation is constrained by the availability of the developers in the case study environment, only a sample use case can be tested for one day. For a real use case in a test setup spanning multiple days, the time and resources are not available.

However, performing the evaluation setup on two different test user groups is possible. Test user group 1 consists of developers of the department in which this MLOps pipeline is developed, and test user group 2 consists of developers from another department. As the developed MLOps pipeline inherently relates to and even builds on existing processes and workflows from the department, the user group of this department should have an advantage in understanding and using the MLOps pipeline. Therefore, it is very beneficial to have a second test user group that is more unfamiliar with the whole process, as this allows to test what happens if the solution is provided to other departments as a blueprint for developing data products.

As only one day is available for each test user group to test the MLOps pipeline, it is not feasible to have them spend most of the day setting up all the necessary infrastructure. Although the developed solution includes the infrastructure setup of the MLOps components, the already implemented MLOps infrastructure is provided to be used in the one-day hackathons. This means the evaluation focuses entirely on using the prepared data product development pipeline to implement an example ML use case.

To utilize the time more effectively, the test user groups are given not only instructions for each use case but also some pre-prepared code snippets for the parts not directly relevant to the process of the MLOps pipeline. Preparing the steps and instructions provided to the user groups in the hackathon in too much detail and with too many prepared snippets harbors the danger of participants only following the instructions and thus deviating from a development process in a real, practical setting.

On the other hand, the time constraints do not allow for the development of the data products from scratch while testing the whole MLOps pipeline. Thus, the test user groups are given the following resources:

- Access to all MLOps platforms and tools used in the pipeline
- Access to a prepared repository containing a template setup of the developed MLOps pipeline
- General documentation on the MLOps pipeline
- Instructions on the use case to implement and tasks to cover the MLOps components during the development of the data product

Their goal is to implement the use case all the way to a finished data product while working through the whole MLOps workflow following the designed pipeline. This setup should allow testing the entire MLOps pipeline and completing the use cases within the limited time frame in a context similar to its intended use.

As two user groups are available for the user tests, two different sample use cases are tested. Both these sample use cases are described below. To avoid unforeseen problems during the user tests about the use cases themselves, they are implemented first as references as part of this thesis. This ensures that any use case related problems have already been resolved beforehand, and during the user tests, there are no major issues, allowing the focus to be on testing the MLOps pipeline and tools. Further details on these tests and the implemented architectures are described in Section 7.1.

Sample Use Case 1: Chatbot answering questions on a climate report

The first use case is creating a chatbot that uses a RAG approach to answer questions about a climate report. The climate report, available as a PDF file, is used to extract text chunks stored with their embeddings in a vector database, providing the knowledge base for the retriever component. In this scenario, no model training is performed; instead, a RAG chain is developed that accesses an existing LLM via API and the retriever component, as described in Section 2.2.1. Since no custom model is being trained this approach deviates somewhat from the classical MLOps workflow. Instead, this use case illustrates how an MLOps pipeline can be applied to more of an LLMOps case (at least regarding using an LLM, not in terms of training an LLM).

Sample Use Case 2: Classification of handwritten digits

The second use case builds on the MNIST[2] dataset to develop a classification model to recognize handwritten digits. This example represents a computer vision use case solved by developing a simple deep learning model. It describes a classical MLOps use case. Based on the MNIST dataset, a model is trained and validated to classify handwritten digits and then integrated into an application to utilize this model.

[2] https://yann.lecun.com/exdb/mnist/

4.3.2 Evaluation Methods

In order to not only enable the developers to test the developed MLOps pipeline, but also evaluate the solution, different qualitative methods of empirical software engineering are applied. Specifically, the method of participant observation [Sea08], an adapted focus group interview [KBL08], and a short anonymous survey are carried out.

Participant Observation

During the evaluation of the MLOps pipeline, which takes place in a hackathon-like setting, the author employs the method of participant observation as outlined by Seaman [Sea08]. This method is selected to gain direct and immediate insights into the practical application of the MLOps solution as users work through a sample use case.

The goal of participant observation in this context is to capture real-time interactions between the users and the MLOps pipeline, focusing mainly on any issues or ambiguities that arise. To document these insights, structured field notes are taken throughout the process, containing the following information:

- Time: When the observation is made
- Context: High-level context, for example, the tool or MLOps component affected
- Description: The details of the observed event
- Involved participants: Just one participant or all participants affected?
- Type of Problem: Documentation issue, technical bug, process problem, ...

To enhance the observational data, participants are encouraged to verbalize their thoughts, actions, and any problems they encounter rather than communicating them through private channels like direct chat, which would be inaccessible to the observer. Inspired by the think aloud observation approach [Sea08], this approach ensures that the observer can gather comprehensive data on the participants' experiences and challenges.

While objectivity in evaluation would ideally require the observer to take a completely passive role and refrain from any intervention, the context of this evaluation necessitates a more flexible approach. Given that the observer is the only person fully familiar with the design, implementation, and documentation of the MLOps pipeline, there may be instances where intervention is required to clarify misunderstandings or address issues stemming from incomplete or unclear documentation. Although the long-term goal is for the MLOps pipeline to function independently without assistance, the initial test scenario represents the first time the solution is being used in practice. It is, therefore, plausible that certain aspects of the documentation are incomplete or unclear. While the test group might be able to resolve these issues on their own, this may lead to unnecessary delays. For the same reason, this may also include providing confirmations or hints regarding minor errors that would eventually be discovered. To ensure that the entire MLOps pipeline can be tested within the limited timeframe of the hackathon, the involvement of the observer is, therefore, allowed if needed. Recognizing that any intervention by the observer could potentially bias the evaluation results, these interventions are minimized and carefully documented. The following details are recorded for each instance of intervention:

- Time: When the observation is made
- Context: High-level context, for example, the tool or MLOps component affected
- Type of intervention: Whether the intervention was a response to a question or an active suggestion by the observer
- Question: The asked question (if applicable)
- Response: The answer to the question or other input given by the observer

Documenting all observations and necessary interventions ensures that the evaluation captures a comprehensive picture of the MLOps pipeline's practical application. Still, not all observations may be equally relevant. However, to avoid distinguishing between potentially relevant or irrelevant observations during the documentation process, every observation, including status updates and completed steps, is recorded. This comprehensive list is later reviewed to identify observations related to problems with the documentation, specific tools, or the MLOps pipeline process itself. These insights gained are essential to refine the current implementation and guide future improvements, ensuring the pipeline can be used in real-world scenarios.

Adapted Focus Group Interview

Following the hackathon, an adapted focus group interview is conducted to gather participants' opinions on the data product development pipeline. This method, based on the method described by Kontio et al. [KBL08], is modified to accommodate the specific constraints of this study. The feedback round is structured around pre-prepared questions using a digital collaboration platform, facilitating efficient and organized feedback collection. Participants are encouraged to discuss their opinions and feedback on the MLOps pipeline within the group setting while simultaneously documenting their feedback on the collaboration platform under the predefined guiding questions. To allow for an open discussion, the prepared questions are intentionally broad, focusing on the following:

- What did you like?
- What did you not like?
- How could it be improved?

This approach fosters open dialogue among participants while ensuring the key points and responses are documented in real time on the collaboration platform. Additionally, a comment section is provided for feedback that does not fit into the predefined categories, allowing participants to share any other relevant insights. However, the setup of the hackathon, where the MLOps pipeline is applied to a prepared sample use case, makes it harder for participants to distinguish feedback regarding the MLOps pipeline from feedback about the test setup prepared for the hackathon. To collect as much feedback as possible, participants are encouraged to provide all input regardless. The feedback is then screened and sorted afterward to focus on the points relating to the MLOps pipeline.

The semi-structured format of this feedback round, combined with the ability for participants to discuss and document multiple points simultaneously, helps to minimize the time required for the session. Furthermore, this method allows for the evaluation of feedback without the need for full interview recording and transcription. To ensure that all verbal discussions, particularly arguments or justifications related to written feedback points, are captured, the feedback sessions are still recorded. However, the recording is only used to review and transcribe specific segments if additional clarification or context is needed for unclear feedback points.

This interview focuses primarily on collecting participants' opinions about the MLOps pipeline, identifying what aspects are well-received or problematic, and, most importantly, gathering suggestions for improvement.

Anonymous Survey

In addition to the open feedback collected through the adapted focus group interview, where participants discuss and potentially influence each other's opinions, a short anonymous survey is conducted to gather individualized input from all participants. This survey is beneficial for answering research question **RQ3**, as it directly addresses participants' opinions regarding simplicity, quality, and speed in developing data products. Additionally, it aims to gather input on the effectiveness of the MLOps pipeline compared to existing processes.

Participants are asked to answer the following statements using a 5-point Likert scale (Strongly Disagree, Disagree, Neutral, Agree, Strongly Agree):

- I found the MLOps pipeline easy to use for developing data products.
- Using the MLOps pipeline significantly reduced the time required to develop a data product.
- I often felt confused or uncertain about how to use certain features of the MLOps pipeline.
- The MLOps pipeline allowed me to focus more on critical aspects of the development process rather than on repetitive or manual tasks.
- The quality of the data product I developed using the MLOps pipeline was higher compared to my previous approaches.
- I found that the MLOps pipeline often failed or encountered errors during critical stages of the development process.
- I would prefer to use this MLOps pipeline for future data product development projects.
- The MLOps pipeline improved my understanding of the MLOps workflow and best practices, which I can apply to other projects.

While many of these insights should surface and be discussed during the feedback session, this survey provides an opportunity for a more structured evaluation of the participants' opinions.

4.3.3 Evaluation Analysis

After collecting feedback and data on the MLOps pipeline through participant observation, concluding feedback sessions, and the anonymous survey, this section describes how these inputs are analyzed to answer research question **RQ3** and generate further insights.

The responses from the feedback sessions are already structured due to the use of the guided template and require only minor processing. Similarly, the anonymous survey results can be easily aggregated and presented. However, the observations and involvements documented by the author during the hackathons need further analysis. For this, the observations are sorted to include those related to encountered problems, and then open coding is applied to these points. This method allows to further categorize and summarize the appeared issues, enabling the identification of key problem themes.

Subsequently, the results from the different methods are joined together to draw conclusions regarding research question **RQ3**. By examining the test of the MLOps pipeline in a practical setting, it is possible to assess the effectiveness of the pipeline when applied to a use case in a real-world similar setup. Specifically, this includes investigating whether the MLOps pipeline enables the test users to develop data products more easily and rapidly.

Furthermore, all additional insights derived through the evaluation are presented. This especially includes suggestions and considerations on what aspects of the designed MLOps pipeline should be adapted or improved, to successfully apply it in an actual project. Overall, the insights gained from this evaluation offer a holistic view of the real-world impact of the developed MLOps pipeline for data product development.

5 Evaluation of Machine Learning Operations approaches

The MLOps field is broad and diverse, with no single solution that fits all scenarios. Instead, a wide range of platforms, tools, and architectural options exists, each addressing specific needs and constraints in data product development. The objective of this chapter is to evaluate these various MLOps approaches, providing a foundation for the tool selection and designing of an effective pipeline for data product development.

In the first section, Section 5.1, this chapter begins by describing the results of a detailed requirement analysis for such an MLOps pipeline. This defines the criteria the developed MLOps pipeline shall meet and thus restricts the solution space of possible MLOps solutions. Additionally, this section describes the results of the performed Rapid Review and the usage of ADRs for documenting the comparisons of the subsequent sections. In order to understand how to best fulfill the defined requirements, a thorough investigation and evaluation of various architectural options and different MLOps platforms and tools is essential. The Section 5.2 begins by identifying the various components of an MLOps architecture and prioritizing them for implementation in the MLOps pipeline. Subsequently, general architectural options, such as different staging concepts and the choice between platforms versus individual tools, are shown in Section 5.3. In the next section, Section 5.4, different MLOps platforms are identified and evaluated. Finally, Section 5.5 describes the result of identifying and analysing tools for each of the previously defined MLOps components.

The insights gained from this comprehensive overview of the current MLOps landscape, highlighting the strengths and limitations of each option, form the basis for the later design of the data product development pipeline in the next chapter, Chapter 6.

5.1 Preliminary

The requirements for an MLOps pipeline can vary significantly depending on the context. Whether developing small regression or classification models, using an existing foundation model via API for a chatbot, or developing complex deep learning models or even custom LLMs, each scenario can lead to specific requirements and varying levels of relevance for different components. Moreover, an MLOps concept is rarely developed from scratch. Existing infrastructure and DevOps processes with corresponding CI/CD tooling are often to be integrated or reused, imposing constraints on the development of the MLOps pipeline.

Given these factors, it is apparent that no universally applicable MLOps solution can be defined. Instead, an initial analysis of requirements must be conducted to be able to develop a tailored solution. Furthermore, it is necessary to get an overview of the platforms and tools in the MLOps field in order to be able to select a suitable solution as well as having a method for preparing, making and documenting all decisions.

Section 5.1.1, therefore, describes the initially conducted requirement analysis, Section 5.1.2 presents the results of the conducted Rapid Review and Section 5.1.3 describes the application of the ADR templates.

5.1.1 Requirement Analysis

This section describes the initially conducted requirement analysis. At first, it includes findings and difficulties encountered during the requirements collection process. Then, the final consolidated requirements including prioritization and justifications are presented. Additionally, the target group is identified, as this can influence the interpretation of the requirements and enable a more customized and better fitting solution.

Findings during Requirement Collection

To gather the different requirements, semi-structured interviews were conducted with multiple team members, including software engineers, data engineers, data scientists, and cloud engineers. This approach allowed for the collection of diverse perspectives and the incorporation of existing experiences. During the collection process it became clear that individual requirements and priorities vary significantly depending on the roles of the interviewees.

For instance, data scientists emphasized the importance of robust experiment tracking, while DevOps specialists prioritized seamless integration with existing DevOps concepts as well as extensive automation. A crucial input from the business side was that while the MLOps solution is to be initially developed and tested within the scope of the case study department, it might be intended to serve as a blueprint and template for developing data products in other areas of the organization, potentially by users with less technical expertise. This highlights the importance of a clear definition of requirements and its intended target user group.

Furthermore, the requirement analysis defines that data access and the potential need to load data from the internal corporate network into cloud storage will not be covered in this thesis. While the evaluation of approaches and tools will consider that some data may be difficult to access from outside the corporate network, the process itself will not be addressed. Instead, the existence of the necessary data for the use of the developed MLOps pipeline is assumed.

Requirements

As described in Section 4.1.1, the requirement collection process was finalized through a collaborative workshop involving the responsible from the case study department as the main stakeholders. This workshop brought together the diverse requirements and established a prioritized list of requirements. The results of this workshop are consolidated in the appendix in Table A.1, which categorizes each

requirement by its type and priority level, already acknowledging that it may not be possible to address all requirements within the scope of this work. Mapped onto the categories of requirements defined in Section 4.1.1, the following overview results:

- **Defined Constraints**: One of the defined constraints is the compatibility of the solution with Microsoft Azure (DC1), which is the predominant cloud provider in the department and across the business unit. Similarly, the continued use of GitHub Actions from the existing DevOps process (DC2) is considered a defined constraint. The last constraint is that the solution shall not incur the need for Kubernetes (DC3), as there is currently no Kubernetes in use and setting up and operating a Kubernetes just for the MLOps solution would require additional knowledge and effort.
- **Functional Requirements**: Any tools used in the MLOps pipeline must be permissible under the company's software usage policies (FR2), meaning only tools with open-source licenses or those already onboarded within the company can be considered. Furthermore, the MLOps solution has to be applicable to all kinds of ML projects (FR1), including those that can only take place on-premise (FR4). Additionally, it is important for the MLOps pipeline to seamlessly integrate into the existing environment and CI/CD concept (FR3).
- **Non-functional Requirements**: The non-functional requirements can, as described in Section 4.1.1, not be defined directly measurable and instead can be see as qualities relevant to the MLOps pipeline. Most highly rated are the maintainability (NFR1) and stability (NFR3) of the solution as well as ensuring that the developed solution offers a good user experience (NFR2) even for less technically experienced users. Further aspects are the performance (NFR4) and modularity (NFR6). Finally it is a factor whether a tool is already used within the company (NFR5). While not directly affecting the evaluation of the approach or tools, leveraging existing tools can provide significant advantages from a business perspective. For example, using tools already onboarded and available as managed services inside the company can significantly reduce the maintenance effort.

Upon closer examination of all the requirements in Table A.1, it becomes evident that fulfilling some requirements can complicate the fulfillment of others. For instance, the criterion for ease of maintenance favors a cloud-based, fully managed solution. However, such a solution may pose challenges for on-premise use cases due to limited access from the internet into the corporate network environment. This trade-off becomes more apparent during the comparison of different MLOps platforms and tools.

Target User Group

An additional critical aspect that is not directly listed among the requirements but is highly relevant for developing the solution is defining the target user group. As mentioned in Section 5.1.1, the solution is designed and tested within the context of the department and a technically proficient team with experience in ML and LLMs. However, the solution should also serve as a blueprint or template for developing data products that could be rolled out to less technically experienced users regarding developing ML projects in other parts of the organization. This means that the target user group for the MLOps pipeline in this work cannot be narrowly defined.

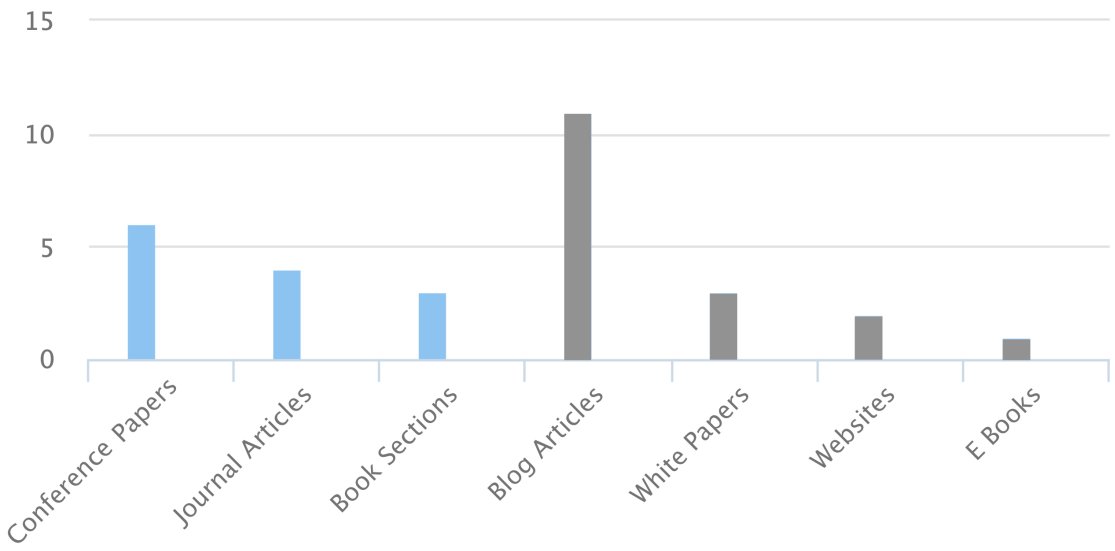


Figure 5.1: Distribution of different types of literature identified in the Rapid Review (white literature types depicted in light blue and grey literature types depicted in grey).

While the primary focus during development is on the needs of the more technically experienced team which also provides most of the requirements, considerations are made to accommodate the needs of the secondary target group wherever possible. Due to their potentially lower technical expertise, this means focusing on a solution that is as user-friendly as possible and can be operated without a high effort and detailed prior knowledge.

5.1.2 Rapid Review Results

After having identified the constraints and requirements for the MLOps pipeline, it is necessary to get an extensive overview about the field of MLOps and state-of-the-art architectures and tools. Therefore, as a foundational knowledge base for all the evaluations in the following sections, a Rapid Review focused on MLOps architecture and tool comparisons was conducted, as described in Section 4.1.

This literature analysis identified 30 relevant sources. As shown in Figure 5.1, 13 sources are categorized as white literature (depicted in light blue) and 17 sources as grey literature (represented in grey). The diagram further illustrates the detailed distribution of the different types of literature. The white literature contains conference papers, journal articles, and book sections. The gray literature is comprised of mainly blog articles as well as white papers, websites and e-books. A detailed overview of all the identified sources can be seen in Appendix B with Table B.1 listing all the white literature, and Table B.2 containing all the grey literature.

5.1.3 Application of Architecture Decision Records

To ensure a structured evaluation of all architecture options and tool comparisons and transparency on the later decisions, ADRs are applied as described in Section 4.1.2. The ADR format created for this thesis can be found in Appendix C. It includes the default problem description, the different options, a decision rationale, as well as a list of participants and the date of the decision.

For the architecture-specific decisions in Section 5.3, general information such as the name and description of the options are included. For the platform comparison in Section 5.4 and tool comparisons in Section 5.5, additional information, such as licensing and cost details, a link to the official tool documentation, and details about the necessary infrastructure and how to set up the tool, are provided. These additional factual details are obtained outside of the Rapid Review sources, typically directly from the tool vendors' websites and their documentation. Furthermore, the ADR for the platform comparison is extended to include an entry that specifically documents which MLOps components are covered or not covered by each platform.

In all cases, the evaluation criteria within the ADR are divided into two dimensions. First, the advantages and disadvantages of different options are derived based on the literature sources from the Rapid Review. Second, the fulfillment of the requirements defined in Section 5.1.1 is examined. This approach ensures that the later decisions made during the design of the MLOps pipeline in Chapter 6 are based on well-founded information and are suitable to the requirements and constraints of the case study environment.

5.2 Definition of Components

As described in Section 2.4, MLOps can be described as a workflow divided into different phases. However, it can also be divided into different MLOps components. In both cases, there is no unique definition, as the field is still developing and is often based on specific cases with different conditions. Even within the sources identified by the Rapid Review, there is no standardized definition of phases or components for an MLOps architecture. With regard to the implementation of an MLOps pipeline with specific tools, it makes sense to structure it by components in order to be able to evaluate and select suitable tools for each component. This section therefore describes the identification and definition of a list of MLOps components, which will be used in the further course of this work. In this step, the defined components are also prioritized within the context of the case study environment to ensure that the evaluation and, more importantly, the design and implementation of the MLOps pipeline focus on the most relevant components. In addition, it is checked whether tools are already in use for individual components, for example specified by the existing DevOps process, and should continue to be used. No further evaluation of potential tools is then necessary for these.

The inclusion criteria for the Rapid Review are not only MLOps architectures, but also tool comparisons, which, as in this work, are mostly structured according to different MLOps components. As a result, the identified literature contains a wide range of information on MLOps components, but also correspondingly many different definitions. This definition is therefore influenced by selected sources that contain a well-founded definition of MLOps components or even phases. Looking at the phases is included as there is a connection between MLOps phases and components. A major influence here is the breakdown of MLOps components from Kreuzberger et al. [KKH23], which

has already been shown in Section 2.4. Notable influence had also the phases of the ML workflow by Databricks [BKTT22]. These consist of *Data Preparation*, *Exploratory Data Analysis*, *Feature Engineering*, *Model Training*, *Model Validation*, *Deployment* and *Monitoring*.

The actual process for determining the different components is closely tied to the process of identifying the MLOps tools, which is explained in more detail in Section 5.5. Ultimately, every mention of an MLOps tool in the identified literature is documented, along with how the tool is categorized within the source and which component or components it serves. This approach not only provides an overview of the available tools but also of the various components. Based on this list and the aforementioned reference models, a unified list of components is created. The entries are sorted into groups, which generally consist of similar but slightly different terms for the same component. The most commonly used term, or a combination of terms that comprehensively covers the component, is selected as the component's name. It is also decided whether a group constitutes a standalone MLOps component. In some cases, especially when there are only a few mentions, it indicates that the component is not commonly listed as a separate component. In these cases, it is checked whether it can be assigned to another component. Additionally, it is assessed whether the grouping truly constitutes a component in an MLOps pipeline. For instance, groupings like *Foundation Model* or *Vector Database* emerge. While these are relevant components, especially for the development of RAG systems, they are application architecture components rather than parts of the MLOps pipeline, and are thus excluded.

Through this process, the literature from the Rapid Review is synthesized into a unified definition of MLOps components. This results in the following list of 13 defined MLOps components, highlighted by italics throughout the remainder of this work:

- *Data Labeling:* Tools for annotating raw data with labels.
- *Data Versioning & Management:* Systems that track and manage dataset versions.
- *Exploratory Data Analysis:* Tools for visualizing and analyzing data trends and making first model experimentations.
- *Feature Stores:* Components for storing and serving ML features.
- *Code Repository:* Version control systems that track and manage code changes.
- *CI/CD:* Tools for automated testing, validation, and deployment.
- *Model Training Pipeline & Workflow Orchestration:* Platforms that automate and manage the model training process.
- *Hyperparameter Tuning:* Tools for automating the search for optimal model hyperparameters.
- *Experiment Tracking & Metadata Store:* Systems that log experiment details for comparison and reproducibility.
- *Model Testing & Validation:* Tools to test and validate models against different metrics.
- *Model Registry:* Components that store and manage model versions, making them available for deployment or rollback.
- *Model Deployment & Serving:* Systems that deploy models for real-time or batch inference.

Component	Priority	Already covered by
Data Labeling	Low	
Data Versioning & Management	High	
Exploratory Data Analysis	Low	Jupyter Notebooks
Feature Stores	Low	
Code Repository	High	GitHub
CI/CD	High	GitHub Actions & Terraform
Model Training Pipeline & Workflow Orchestration	Medium	
Hyperparameter Tuning	Medium	
Experiment Tracking & Metadata Store	High	
Model Testing & Validation	High	
Model Registry	High	
Model Deployment & Serving	High	
Monitoring & Observability	High	

Table 5.1: Overview of defined MLOps components with priorities and if they are already covered by a tool.

- *Monitoring & Observability:* Tools to monitor model performance, detecting drift or degradation in production.

After the identification of the components, an assessment is conducted to determine the extent to which the existing DevOps framework within the case study environment already covers certain MLOps components. As shown in Table 5.1, this is the case for some instances: For the *Exploratory Data Analysis* component, Jupyter Notebooks are already in use and will continue to be utilized. Furthermore, the knowledge base from the Rapid Review does not suggest any additional available tools for this component. Additionally, the existing DevOps setup already incorporates GitHub with GitHub Actions and Terraform for the *Code Repository* and *CI/CD* components, which are also set to continue being used. Consequently, these components will not be further evaluated in the subsequent stages.

Finally, a prioritization of the components was conducted in collaboration with the team, which is also included in Table 5.1. This prioritization influences the design of the MLOps pipeline and is especially relevant for the implementation phase, ensuring that the most critical components are implemented first. In general, most components are assigned a high or medium priority. Among the components to be evaluated, only *Data Labeling* and *Feature Stores* are considered to have a low priority. This is primarily because no current or upcoming ML projects are expected to require *Data Labeling*, and extensive data access and data preparation for the development of features and storage in feature stores are not currently available within the case study environment. Of the remaining components to be evaluated, the *Model Training Pipeline & Workflow Orchestration* and *Hyperparameter Tuning* component are categorized as a medium priority. This is motivated by

the current focus on Gen AI within the department, for example developing chatbots with RAG approaches. In this setup existing LLMs are used because it is not feasible to develop own LLMs, primarily for cost reasons. As a result, the necessity for an extensive training infrastructure with hyperparameter tuning of models is currently not as high. All the other components are labeled with a high priority as they are seen as the most important and necessary components for not only developing but also operating data products.

5.3 Architecture Options

After having defined the different components an MLOps pipeline could include, there is still the question on how to achieve the functionality of these components. While this ultimately depends on the selection of specific tool, it first brings up a broader question: whether to use an MLOps platform, some of them even promising to cover the complete end-to-end MLOps process or using rather single specialized tools for each component and putting them together to form a customized MLOps pipeline. Another question is how the whole process of ML model development is to be included into a possibly existing DevOps workflow and environment staging concept. Basically, how the integration of the different MLOps components into one smooth MLOps workflow can be achieved for developing mature data products merging model development with application development.

Platform versus Tools

The decision between using an MLOps platform or individual tools is addressed in several sources from the Rapid Review. For example, Symeonidis et al. [SNKP22] describe the differences and advantages of both options. Additionally, a third option emerges in the literature: using a hybrid approach, where an MLOps platform is extended by additional specialized tools. These three architectural options are documented and evaluated in relation to the requirements in an ADR. The detailed evaluation is available in the supporting GitHub repository[1]. The summary of the architectural options is as follows:

Single platform: Utilizing a single platform provides a simple and unified approach where users only need to learn one system. This option is generally reliable and stable, with excellent integration between components, enhancing overall performance and user experience. Maintenance is more straightforward since there's only one platform to manage. However, this approach has limited flexibility; adding new features can be difficult if the platform doesn't already support them. The lack of modularity and the danger of vendor lock-in means the solution may not adapt well to future needs or changing requirements.

[1] https://github.com/gitpold/master-thesis-mlops/blob/v1.0.0/mlops-comparison/architecture/platform-versus-tools.png

Platform extended by tools: Extending a platform with additional tools combines the benefits of a unified system with the flexibility of specialized extensions. This approach is easier to maintain compared to fully modular solutions, as it involves managing one main platform alongside a few tools. It is more adaptable to unique use cases while maintaining consistent user experience and integration. On the downside, it still requires managing compatibility between the platform and the extending tools. Users may find it slightly more complex, needing to learn both the main platform and the additional tools.

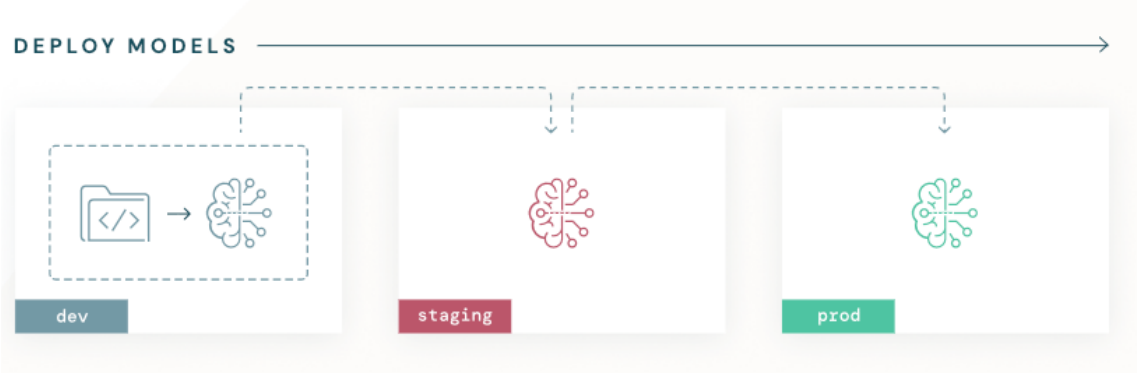
Combination of single tools: Combining individual tools offers maximum flexibility, allowing the selection of the best tool for each component. This high modularity enables components to be reused or replaced independently, promoting adaptability. However, integration between these tools can be complex and potentially unreliable, leading to increased development effort. This option entails a higher maintenance overhead since more tools need to be managed separately. Additionally, it requires users to learn multiple different tools, which can complicate the user experience and steepen the learning curve.

Findings: A platform extended by single tools offers a good balance between maintainability and flexibility. It simplifies operations by primarily relying on one platform while allowing specialized tools to fill gaps in functionality. In contrast, using a single platform limits adaptability, and assembling multiple tools increases complexity and maintenance effort.

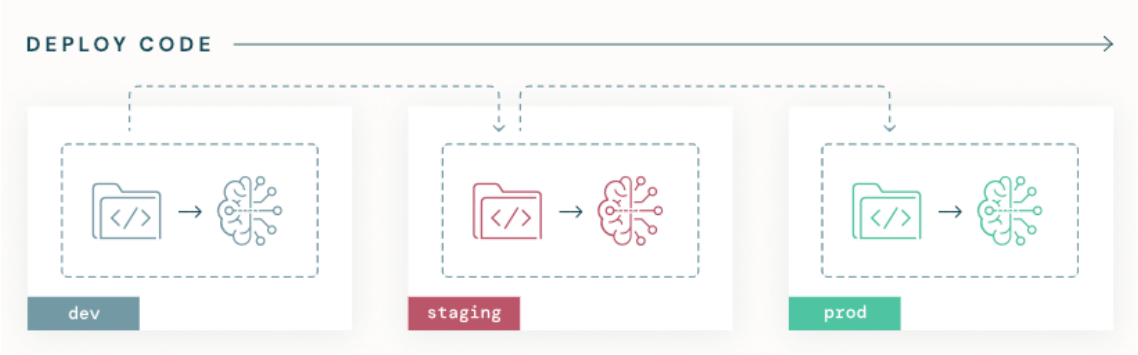
Model Staging

Model staging refers to the process of managing ML models as they progress through various stages. It ensures that models are properly tested before reaching a production environment. In relation to model deployment and staging, two different concepts have been identified in the sources from the Rapid Review. Databricks, for instance, names these two concepts the *Deploy Models* (Figure 5.2a) and *Deploy Code* (Figure 5.2b) approaches as seen in Figure 5.2 [BKTT22]. The *Deploy Models* option trains the model once on a `DEV` stage and then promotes the trained model to the next stages. Instead, the *Deploy Code* approach propagates not the model, but the model training code through the stages up to the production environment and trains the model in each stage separately. The following summary of the ADR outlines the most relevant differences and advantages of both approaches.

Deploy Models: This approach offers an easier handoff for data scientists, as they can focus on model development without worrying about production complexities. It reduces costs associated with model training, which is particularly beneficial for expensive models like deep learning architectures and large language models (LLMs). However, this method has drawbacks. It may not be feasible if production data isn't accessible from the development environment, limiting its applicability. There's also no support for automated model retraining, which means the model



(a) Deploy Models



(b) Deploy Code

Figure 5.2: Different ML model staging/deployment concepts [BKTT22].

may become outdated over time. Additionally, this approach still requires a separate code deployment path for production, adding to the maintenance overhead.

Deploy Code: This option enables better access control and supports automated model retraining, ensuring the model remains up-to-date with the latest data. It integrates seamlessly with existing continuous integration and continuous deployment (CI/CD) processes, and any ancillary code—such as preprocessing or post-processing scripts—follows the same staging as the model training code. This results in a more stable process that's scalable across multiple projects. However, deploying the code presents challenges. It introduces a steeper learning curve for data scientists who must manage code across different environments. There's also the potential for increased costs when retraining large models in production due to the computational resources required.

Findings: The *Deploy Code* approach aligns well with CI/CD workflows, offering better scalability and stability, though it requires more effort to manage training code. The *Deploy Models* approach is simpler and potentially more cost-effective for expensive models but lacks flexibility. The theoretical evaluation suggest *Deploy Code* may be more suitable for multiple projects and integrates easier with existing staging processes for application code.

Staging Concept

Beyond the model development and staging process, a complete staging concept for a data product contains additional details that have to be decided upon. For instance, a DEV stage could be exclusively used for model development but might not involve further testing or deployment of a model, while a PROD stage could include complete integration tests, automated model deployment, and potentially even automated retraining. In the literature of the Rapid Review various staging concepts have been identified that differ in these details.

In the case study environment, an existing staging concept for application development is already in place as part of the DevOps framework. This includes the typical three environments, here labeled DEV, QA, and PROD. While there are minor differences between these stages, the general approach is that all three stages contain the same components. This setup allows for the complete application, along with all its dependencies, to be developed and tested on DEV. The QA version can then be used for letting some users test new features, while PROD runs the stable, tested version of the application.

Of course, the development process for web applications differs from that of ML models. However, if the *Deploy Code* approach is used for ML model development, these processes become more similar. A potential integration of both development cycles would mean that a model would need to be not only trained but also deployed on DEV in order to be consumed by the application in that stage. Alternatively, the ML model development process could be treated completely separate, using its own staging environments. In this case, a deployed PROD version of the ML model could be used in the DEV stage of the application development. This would allow the development team to focus on application development and testing, knowing the model has already been validated in its own development cycle. However, for a data product where the main functionality depends on the interaction between the model and the application, it might be relevant to test their interaction directly. This shows, that a key decision when designing a development pipeline for data products is how to integrate and whether to synchronize the ML and application development cycles.

5.4 Evaluation of Platforms

Building on the insights gained from the comparison of architectural options in Section 5.3, it became evident that utilizing an MLOps platform is a feasible approach for this case study environment and its requirements. This section delves into the identification and comparison of various MLOps platforms based on the literature gathered through the Rapid Review.

5.4.1 Identification of Platforms

To identify the platforms mentioned in the literature, a process similar to the one in Section 5.3 has been applied. Just as each tool has been mapped to a specific component, all mentions of MLOps platforms in the Rapid Review sources have been systematically collected. This creates a comprehensive list of potential platforms, even allowing to rank them based on the frequency of mentions, which gives an indication of their prevalence in the field.

One challenge that emerged is the inconsistency in how different sources classify certain technologies. Some sources defined technologies as MLOps platforms, while others classify them as tools that cover one or even several MLOps components. A prime example of this classification challenges is MLflow, which was identified 8 times as a platform and 14 times in the context of individual components. In 10 cases, MLflow was linked to the *Experiment Tracking & Metadata Store* component, but it was also mentioned in connection with the components *Model Registry* and *Monitoring & Observability*. This amount and distribution of mentions suggests that MLflow is an established technology with a primary focus on experiment tracking, while also covering some other essential components of the MLOps workflow. However, it is debatable whether MLflow qualifies as a complete MLOps platform, as it apparently lacks full end-to-end MLOps lifecycle support. For the purposes of this thesis, MLflow is compared in the platform category. Additionally, however, its core functionalities are explored further in Section 5.5 when comparing tools for the individual MLOps components.

List of Platforms

The complete list of platforms derived from the Rapid Review resulted in the designation of a considerable number of platforms. Specifically, over 20 platforms have been identified that are not only named as such, but are also not obviously misclassified. This led to a list of platforms as shown in Appendix D, that was too extensive to be closer evaluated in the scope of this work. In order to reduce the number of MLOps platforms for a more precise comparison, the following criteria were applied:

- Frequency of mentions in literature: Platforms that were named more often in the Rapid Review were prioritized.
- Author's assessment: Based on personal experience from performing the Rapid Review and initial research, the author has prior knowledge about potentially promising platforms, which was incorporated.
- Existing comparisons: Platforms that have not only been mentioned but already been compared or evaluated in the literature of the Rapid Review were prioritized.
- Case study relevance: Platforms that already have been used within the case study environment were preferred as to potentially reduce the learning curve.
- Team input: Feedback and experiences from the team involved in the case study were factored in.
- Requirements: The platforms were already filtered based on their compatibility with the defined requirements. For example, although the requirement Azure compatibility does not exclude solutions from direct competitors like Amazon Web Services or Google Cloud Platform, they are deprioritized due to potential integration and user experience challenges.
- Technology variation: It was made sure to consider a mix of commercial platforms by cloud providers or designated MLOps companies, as well as open-source solutions.

Based on these criteria, the following MLOps platforms were selected for further comparison. All these platforms are indicated by the use of small caps throughout this thesis.

- Azure ML
- Databricks
- MLflow
- Kubeflow
- Iguazio
- Weights & Biases

5.4.2 Comparison of Platforms

The following provides a high-level overview of each platform and summarizes the evaluation results. A detailed comparison in an ADR as described in Section 5.1.3 can be found on GitHub[2].

Platforms

Azure ML: Azure ML offers an extensive, user-friendly platform with strong integration into the Azure ecosystem, making it a reliable and scalable solution. However, its limited support for on-premise use with the limitations in the case study environment and deep dependency on Azure may reduce flexibility for multi-cloud or hybrid setups.

Databricks: Databricks provides multi-cloud support, robust data engineering capabilities, and integration with Azure, along with an open-source focus. Its managed MLflow integration is valuable, but familiarity with Spark is essential. Given its existing use in the team and strategic alignment, Databricks presents itself as a strong option for this case study.

MLflow: MLflow is an open-source, modular platform that excels in experiment tracking and model registry. However, it lacks the full suite of end-to-end MLOps components, requiring additional tools and increasing maintenance efforts.

Kubeflow: Kubeflow is highly scalable and modular, with strong Kubernetes-based workflows. However, its complexity and the expertise required to manage it make it a less ideal fit for this use case.

Iguazio: Iguazio provides a fully integrated platform with flexible deployment options, including multi-cloud and on-premise setups. However, the platform's complexity and the onboarding process might make it less feasible for this case study, especially considering the time and effort required for setup.

[2] https://github.com/gitpold/master-thesis-mlops/tree/v1.0.0/mlops-comparison/platforms

Weights & Biases: Weights & Biases offers strong experiment tracking and model management but requires an onboarding into the company that isn't practical for this thesis. It also lacks some components, necessitating additional tools for a complete MLOps solution, complicating integration and maintenance.

Findings: The comparison reveals that managed platforms seem to provide a good balance of ease of use and flexibility. In particular, Azure ML and Databricks emerge as the strongest candidates. Both platforms cover most essential MLOps components and align well with the defined requirements, only lacking in support for on-premise use cases.

5.4.3 Summary

Based on the literature of the Rapid Review a multitude of MLOps platforms were identified. Even if not all of them could be evaluated in depth, the most relevant ones were compared in more detail. In this process, Azure ML and Databricks were identified as promising candidates. The Gartner report *Magic Quadrant for Data Science and Machine Learning Platforms*[3] also identified these two as leaders in the field of MLOps platforms. In this initial comparison, both platforms performed well, covering nearly all the essential features and relevant components. The only significant lack lies in the limited support for on-premise use cases, but since this requirement was assigned a lower priority, it doesn't weigh as heavily. Both platforms offer extensive user interfaces that simplify operation, and since they are managed services, they can be configured either through Terraform (Infrastructure as Code (IaC)) or directly in the Azure portal, keeping deployment and maintenance effort lower than if everything has to be done by hand. Ultimately, both platforms fulfill the majority of requirements for the case study environment. For the final choice, addressed as part of the MLOps pipeline design in Chapter 6, it might make sense to test the setup of both platforms in practice or include further evaluation criteria.

5.5 Evaluation of Tools

After having evaluated different MLOps platforms, one key finding is that many so-called platforms do not provide a complete, end-to-end MLOps experience. Often, they focus on a few components while lacking full coverage of others. This led to the conclusion that even when using an MLOps platform, it may still be preferable, or even necessary, to use specialized tools for certain components. Therefore, a tool comparison was carried out for the components defined in Section 5.2. However, only those components for which no tools were already specified (e.g., as part of the existing DevOps process) as outlined in Table 5.1, have been evaluated.

5.5.1 Identification of Tools

The process for identifying relevant tools is similar to the identification of MLOps platforms in Section 5.4 and the identification of components in Section 5.2. Again, the tools were identified based on the literature from the Rapid Review. As before, each source was reviewed, and any

[3] https://www.gartner.com/en/documents/5509595

mentioned tools were collected and mapped to their respective component or components. This process resulted in a comprehensive list of tools and technologies in the MLOps landscape, along with information on how frequently each technology is mentioned and in the context of which MLOps components it occurs.

As seen with the identification of platforms, and hinted at during the component identification process, the distinction of tools and components as well as the mapping of tools to components is not always clear-cut. This ambiguity arises because different sources may categorize the same tool differently. Additionally, some tools cannot be strictly mapped to just one component, as they sometimes cover multiple components. Another difficulty in mapping tools to components arose from the differences between open-source and premium versions of the same tool. Many providers offer both a free, open-source version and a paid, cloud-based variant, which often includes more features or evolved into a managed service with additional capabilities. It is often unclear from the Rapid Review sources whether they are referring to the open-source tool or the managed premium version, which can lead to errors in the description of licenses, costs, and available features.

For the concrete mapping of tools to their corresponding components, each tool was documented along with the component description it appeared with in the sources. Each tool was then assigned to its respective MLOps component as defined in Section 5.2 or it was marked as irrelevant if it did not belong to the MLOps lifecycle (for example a vector database or LLM). In unclear cases, the tool's website was consulted to obtain more information and ensure proper categorization.

As with the platform evaluation, some tools were frequently mentioned across multiple sources, while others appeared only once. And as in the platform comparison, there were components for which too many potential tools exist to be evaluated meaningfully within the given time constraints of this thesis. Therefore, a further sub-selection of tools was made for the components applying the same criteria as for the sub-selection of MLOps platforms as described in Section 5.4.

This approach ensured that newer and less frequently mentioned tools were not categorically excluded but were also considered. In rare cases this even led to tools that did not appear in the Rapid Review being included in the evaluation. Additionally, if relevant, single functionalities of MLOps platforms were also considered in the comparison for a component. This was especially done for Azure ML and Databricks, which both emerged as promising candidates from the platform evaluation in Section 5.4.

5.5.2 Comparison of Tools

The following overview present the results of the tool comparisons for each MLOps component. The focus is on the key insights gained from comparing the tools, highlighting which tools are potential candidates for being used in the MLOps pipeline and which are excluded from further considerations due to specific limitations. The detailed comparisons in the form of ADRs as mentioned in Section 5.1.3 can be viewed on GitHub[4]. Like the platforms, all tools are highlighted in small caps throughout this thesis.

[4] https://github.com/gitpold/master-thesis-mlops/tree/v1.0.0/mlops-comparison/tools

Component: *Data Labeling*

Labelbox: A comprehensive and collaborative data labeling platform supporting various data types (text, image, video). It is a managed services, making it low maintenance and scalable, but it's a proprietary, paid tool.

Scale: Combines human annotators with AI for efficient, scalable labeling, supporting a variety of data types. While also a managed service, it is a proprietary, paid service.

Snorkel Flow: Focuses on programmatic labeling and flexible label function creation. While reliable and scalable, there is no clear documentation. And while offering an open-source library, this version is proprietary and paid.

Doccano: A free, open-source tool focused only on text labeling. It can be run locally or deployed, making it flexible but limited to text tasks, lacking support for other data types.

Azure ML Data Labeling: Integrated into the Azure ML platform, offering a full range of features for different data types. However, using it for just data labeling without the full Azure ML suite may be inefficient.

Findings: Labelbox and Doccano could be potential candidates based on the individual project requirements, while Scale and Snorkel Flow would need additional onboarding efforts. Azure ML Data Labeling would be a natural option in case Azure ML is used as MLOps platform.

Component: *Data Versioning & Management*

DVC: A lightweight, command-line tool for data versioning integrated with Git. It is easy to use and versatile, supporting all cloud platforms and storage types. While low-maintenance, it may have performance issues with large datasets.

Pachyderm: A powerful data versioning and pipeline orchestration platform. It offers great scalability, but its reliance on Kubernetes adds complexity and maintenance effort.

Delta Lake: Provides transactions and metadata management on large-scale data. It integrates well with Spark for high performance but is more complex to use due to its dependencies.

LakeFS: Offers Git-like version control for data lakes with easy support for large datasets and CI/CD integration. While scalable, it requires more setup effort compared to managed services like Azure ML or Databricks.

Azure ML Data Versioning: Integrated with Azure ML, this managed service offers seamless data versioning with low maintenance, but lacks modularity, making it harder to integrate with other tools.

Databricks Data Versioning: Built on Delta Lake, it provides high scalability and reliability as part of the Databricks platform.

Findings: Pachyderm is eliminated due to its dependence on Kubernetes, leading to higher maintenance. While DVC and LakeFS are valid solutions, their advantages over the versioning capabilities already provided by Azure ML or Databricks are limited. Delta Lake on its own and its dependency on Spark seems complex, but the integrated Delta Lake in case of choosing Databricks would be a suitable solution. Similarly, the data versioning in Azure ML is a valid solution in case Azure ML is used as a platform.

Component: *Feature Stores*

Feast: An open-source, cloud-agnostic feature store that supports both online and offline storage and real-time feature serving. It's easy to set up locally but may require some effort for deployment of an user interface.

Tecton: A managed service feature store that simplifies feature management with an intuitive User Interface (UI). Designed for enterprise-grade reliability, it's scalable but requires cloud infrastructure. Documentation contains no instructions for connecting to Azure.

Hopswork: A comprehensive platform offering feature engineering, storage, and serving, with enterprise-grade reliability. It can be deployed on-premise or on the cloud.

Featureform: A flexible and infrastructure-agnostic feature store, designed for ease of use with intuitive UI.

Findings: Based on the theoretical comparison there is no clear best tool identified. As this component has only been assigned a low priority this is viable.

Component: *Model Training Pipeline & Workflow Orchestration*

Apache Airflow: A mature workflow orchestration tool that supports various platforms like Google Cloud Platform, Amazon Web Services, and Azure. It offers a flexible UI and is scalable but requires setup and maintenance efforts.

Argo Workflows: Argo Workflows is a Kubernetes-native tool designed for complex workflows. It's lightweight and highly scalable but requires knowledge of Kubernetes, which increases setup complexity.

Flyte: Provides scalability and flexibility with strong multi-language support and UI features. However, it requires a Kubernetes cluster, adding maintenance overhead.

Kedro: Kedro is a Python framework for building pipelines, focusing on reproducibility and collaboration. It integrates with other orchestration tools but lacks orchestration capabilities itself.

Kubeflow Pipelines: Tailored for ML workflows on Kubernetes, providing scalability and tight integration with other Kubeflow components. However, it requires Kubernetes knowledge and setup.

Prefect: Prefect offers a modern UI and lightweight orchestration but also depends on Kubernetes for large-scale deployments, which adds complexity.

Azure ML Jobs & Pipelines: A managed service with low maintenance, offering a visual approach to creating pipelines. It's tightly integrated with Azure and scalable but lacks on-premise support.

Databricks Jobs & Workflows: A managed service with a code-centric approach, offering highly scalable solutions using Spark, but like Azure ML, lacks direct on-premise resource access.

Findings: Argo Workflows, Flyte, Kubeflow Pipelines, and Prefect require Kubernetes, making them less appealing due to higher maintenance efforts. Kedro, while useful for pipeline authoring, lacks orchestration functionality on its own. Airflow could be a viable option if its complexity can be managed, particularly via a company internal managed service. Azure ML and Databricks provide seamless, scalable options, but neither supports on-premise resources.

Component: *Hyperparameter Tuning*

Optuna: An open-source framework for hyperparameter optimization, offering various algorithms, distributed optimization, and CI/CD integration. It's easy to maintain and scales well in distributed setups.

Ray Tune: A Python library for hyperparameter tuning within the Ray ecosystem. It excels in distributed optimization, handles large datasets, and scales effectively through Ray clusters. It integrates smoothly with CI/CD pipelines.

Hyperopt: A simple Python library for distributed hyperparameter optimization, suited for complex configurations but offering limited scalability compared to Ray.

Findings: Based on a theoretical comparison no identification of a single best tool can be achieved. Instead the most suitable tool can be chosen based on specific project requirements as there are no major dependencies to other components. This is valid, as this MLOps component is only categorized by a medium priority.

Component: *Experiment Tracking & Metadata Store*

Aim:	An open-source metadata tracker with an interactive UI. It handles large volumes of tracked data but may not scale well for extensive experiments.
CML:	A tool for integrating CI/CD into ML, making it ideal for tracking experiments within CI pipelines.
Guild AI:	A command line based toolkit for experiment tracking and optimization, with good flexibility but a steeper learning curve.
TensorBoard:	It is primarily suited for TensorFlow, offering visualization for ML experiments.
MLflow Tracking:	A popular tool for logging parameters, metrics, and outputs across multiple ML frameworks.
Azure ML Experiments:	Offers a managed service for experiment tracking with built-in MLflow compatibility, reducing maintenance overhead.
Databricks Experiments:	Offers a fully managed MLflow integrated into the Databricks platform.

Findings: TensorBoard is excluded as it mainly supports TensorFlow. MLflow is widely used, but both Azure ML and Databricks offer integrated MLflow-compatible experiment tracking, making a standalone MLflow unnecessary in case these platforms are used. For on-premise cases where Azure ML or Databricks can't be used, an on-premise MLflow could be a suitable solution. Aim and Guild AI provide local experiment tracking but offer no clear advantages over MLflow and thus Databricks. CML focuses on CI pipeline results and is not directly relevant, though it could enhance future CI processes.

Component: *Model Testing & Validation*

Alibi Explain:	Python library for model inspection and explainability. Primarily useful for classification and regression models.
Arize Phoenix:	Observability and evaluation tool for LLMs, providing built-in metrics and monitoring capabilities. Suitable for LLM/RAG evaluation, though full functionality requires paid Arize service.
Deepchecks:	Comprehensive validation tool for classical ML models with a paid LLM extension. Low infrastructure overhead, but more suited for testing in development phases.
Fiddler:	Enterprise AI observability tool, requiring complex Kubernetes setup and paid license, which limits usability.
Great Expectations:	Tool for data validation, profiling, and testing. Does not focus on model testing, making it less relevant for this context.

Giskard: Holistic testing platform covering classical ML and LLMs/RAG systems. Minimal setup with broad functionality makes it ideal for CI/CD integration.

Galileo: Gen AI observability stack. Lack of cloud onboarding in company and complex setup for self-hosting makes it less viable.

Ragas: Python library specialized in RAG evaluation. Provides essential metrics and can integrate with tools like Giskard and Arize for more comprehensive validation.

Findings: Fiddler and Galileo are excluded due to lack of cloud onboarding and complex on-premise installation needs. Great Expectations focuses on data, not model testing, and is not considered further. Alibi Explain offers model explainability but is not vital for general setup. Deepchecks supports classical ML models but has additional complexity for LLM evaluation and production monitoring. Giskard seems like a viable solution for both classical ML and LLM/RAG sytems with no deployment concerns. Ragas could provide an additional validation tool when developing RAG systems.

Component: *Model Registry*

MLflow Model Registry: Open-source, flexible model registry for local, on-premise, or cloud setups. Easy to use with good community support, but lacks multi-tenancy.

Kubeflow Model Registry: Part of the Kubeflow ecosystem, scalable but requires Kubernetes expertise and can be complex to manage.

Azure ML Model Registry: Managed service integrated with Azure ML, offering low maintenance and strong integration with Azure DevOps and GitHub.

Databricks Model Registry: Managed MLflow registry within Databricks, offering a user-friendly interface and easy possibility to leverage all advantages of MLflow.

Findings: The Kubeflow Model Registry is dropped because of the Kubernetes dependency. MLflow provides a state-of-the-art model registry but does not support multi-tenancy. With Azure ML and Databricks being promising candidates to be used as an MLOps pipeline, either the MLflow compatible Azure ML model registry or the model registry in Databricks, which is exactly MLflow, would be suitable choices.

Component: *Model Deployment & Serving*

Seldon Core: Offers powerful model deployment but has a complex setup and Kubernetes dependency, making it less suitable.

KServe (Kubeflow): Highly scalable and integrates well with Kubernetes but requires Kubernetes expertise, which complicates its use.

BentoML: Excels at packaging models into Docker but does not provide direct model serving.

LangServe: Great for deploying language models but not a general model serving tool, making it suitable for specific use cases only.

Ray Serve: Provides scalable serving and on-premise deployment but can be complex to integrate.

Azure ML Endpoints: Offers seamless integration with Azure ML and is low maintenance.

Databricks Serving: Offers model serving within the Databricks platform.

Findings: Azure ML and Databricks both offer comprehensive model serving features that align well with their respective platforms, making them ideal choices for integrated model deployment. KServe and Seldon Core are excluded due to their reliance on Kubernetes and licensing uncertainties. Ray Serve, while suitable for on-premise deployments, is less favorable compared to the cloud-based solutions provided by Azure ML and Databricks. BentoML's role in packaging models into Docker containers is less relevant given the direct serving capabilities of Azure ML and Databricks. LangServe is considered useful for specific LLM/RAG applications but does not serve as a general model serving platform.

Component: *Monitoring & Observability*

Arize AI: A comprehensive observability tool for LLMs but is costly and complex to deploy.

Evidently AI: Supports tabular and text data and is easy to use but lacks broader data support.

Prometheus + Grafana: Offers robust monitoring but requires extensive setup and maintenance.

Fiddler: Provides enterprise-level observability but is complex and proprietary.

Helicone: Focuses on LLM observability with many features but has a complex deployment process.

Langfuse: Ideal for LLMs with straightforward deployment, though it may face scalability issues.

Azure ML Monitoring: Well integrated model monitoring solutions for classical model development

Findings: Prometheus and Grafana, though robust, involves too much setup and maintenance in general and for model-specific monitoring. Arize AI and Fiddler are excluded due to their complex deployment and licensing. Evidently AI is limited to text data, making it less suitable as a general solution. Helicone, despite its broad feature set for LLMs, has a high deployment effort and is not focused on visualizing LLM evaluations. Langfuse is effective for LLM monitoring with a simple deployment process. Azure ML Monitoring or potentially even the not compared Databricks provide well-integrated functionality of modeling and observation.

5.5.3 Summary

In summary, many components can effectively be covered by functionalities already integrated into platforms like Azure ML or Databricks. However, in some cases, such as monitoring LLMs responses, specialized tools like Langfuse or Arize Phoenix offered at least during the time of writing of this thesis distinct advantages over both platforms, making their additional use worthwhile. It also became evident that making a conclusive decision based solely on a theoretical evaluation can be challenging. For components with few dependencies to other components, the best tool may even depend heavily on specific project requirements and cannot be universally defined. For instance, tools suitable for text data labeling can be different to those appropriate for video data labeling.

As a result, while promising candidates for each component have been identified for a potential use in the MLOps pipeline, the final decisions remain open. This is also because the interplay between different tools is not explored yet, which also influences the final selection. Most of these decisions are addressed in the upcoming chapter, Chapter 6, about designing the MLOps pipeline where practical insights and compatibility of tools are examined to make these final choices. Some component decisions, however, are only narrowed down to promising tools but are kept open to be decided based on individual project requirements.

6 Design and Implementation of a Data Product Development Pipeline

This chapter focuses on the design and implementation of an MLOps pipeline tailored to the development of data products as described in Goal 2. Building upon the insights gained during the evaluation of MLOps approaches in the previous Chapter 5, a solid understanding of architectural strategies, potential platforms, and tools has been reached. Additionally, the requirements collected in Section 5.1.1 are integrated into the design to develop a targeted MLOps pipeline that aligns with the specific requirements of the case study environment. Afterwards, the effectiveness of the developed MLOps pipeline can be assessed, as described in the subsequent Chapter 7.

In Section 6.1, the process of designing and developing the MLOps pipeline using the Design Science methodology [Wie14] is shown. Additionally, the challenges encountered during the implementation process and how their resolutions influenced the overall design are presented.

Subsequently, the finalized MLOps pipeline is presented in Section 6.2. It includes the MLOps infrastructure, encompassing the platforms and tools for each MLOps component, as well as a detailed description of the workflow for using the MLOps pipeline. Furthermore, a template repository and instructions are presented, serving as a starting point for the development of data products with this MLOps pipeline.

6.1 Design Process

The design process includes the implementation of the necessary infrastructure and follows the design cycle defined by Wieringa [Wie14], as described in Section 4.2. This section outlines the design process, adhering to the tasks of the design cycle: *problem investigation*, *treatment design*, and *treatment validation*. The outcome of this design problem, referred to as artifact in Design Science, is the concept of an MLOps pipeline, including the required infrastructure setup. The progression through the tasks of the design cycle did not occur strictly sequentially and, for the sake of clarity, is also not presented that way in this work.

Section 6.1.1 still begins with the *problem investigation* task, again stating the specific problem to be solved or improved. The subsequent task, *treatment design*, is detailed in Section 6.1.2 and shortly describes the design process of the MLOps pipeline. In regard to the design itself, first the selection of an MLOps platform is covered in Section 6.1.3 and then the selection of tools for each MLOps component in Section 6.1.4. Finally, in Section 6.1.6, the complete MLOps pipeline is validated not only against the defined requirements but also in terms of the feasibility in regard to practical implementation pf the components.

6.1.1 Problem Investigation

In this case, the underlying problem to be addressed closely aligns with the overall objective of this thesis: to simplify and accelerate the development of data products with an MLOps pipeline. The specific requirements for the design are already outlined in Section 5.1.1 in detail. As mentioned there are the defined constraints for the solution to be compatible with Azure (DC1) and GitHub (DC2), and to work without the need of a Kubernetes platform (DC3). Further there are the functional requirements of being permissible within the company from a licensing standpoint (FR2), integrating with the existing DevOps concept (FR3) and supporting various types of ML projects, including classification or regression models, deep learning, and applications involving LLMs (FR1). The originally defined requirement to support on-premise use cases (FR4) was already assigned a low priority from the beginning and, in consultation with the stakeholders, is considered here unlikely to be achievable due to other, more important requirements. However, it became clear that a full onboarding of tools not yet in use within the company is not feasible within the scope of this work due to time constraints; therefore, platforms and tools must be selected accordingly. Of the non-functional requirements the ones with high priority, namely maintainability (NFR1), stability (NFR3), and user experience (NFR2), are to be considered foremost.

The overall stakeholders' goals can be summarized as twofold. Firstly, to streamline the development of data products within the department. Secondly, to provide this solution as a blueprint and starting point for other departments within the business unit, thereby enabling them to independently develop their own data products. As discussed in Section 5.1.1, this means there is no single, clearly defined target user group. Instead, there are two user groups, with the designed solution mainly tailoring to the needs of the developers from the case study department, but ideally also accommodating a target user group less familiar with the development of ML models and data products.

6.1.2 Treatment Design

After having identified the problem to be addressed and defining the stakeholder goals in the previous step, the *treatment design* task contains clearly formulating the requirements for the solution. In summary, the goal is to design an MLOps pipeline that fulfills the requirements of the case study environment as described in Section 5.1.1 and summarized in Section 6.1.1, thereby simplifying and accelerating the development of data products. Specifically, the solution shall provide coverage for each of the MLOps components defined in Section 5.2, either through a platform or a tool. The prioritization of these components and pre-existing tools were also considered, focusing on high-priority components and not modifying existing tools. Special attention was paid to ensuring that all selected tools are compatible with these existing components and between each other. This led to the following list of components to be considered during the design, with some of them being categorized as optional based on their initial prioritization and weak dependencies to other components.

- *Data Labeling* (optional)
- *Data Versioning & Management*
- *Feature Stores* (optional)
- *Model Training Pipeline & Workflow Orchestration*

- *Hyperparameter Tuning* (optional)
- *Experiment Tracking & Metadata Store*
- *Model Testing & Validation*
- *Model Registry*
- *Model Deployment & Serving*
- *Monitoring & Observability*

In this step, multiple artifacts are developed to potentially solve the identified problem. Since both the design cycle and the MLOps pipeline's individual component decisions influence each other, this design process is not only iterative but also highly depending on its earlier decisions. Therefore, the following shows not the complete designs, but the single decisions as they are building on each other. Each decision includes the options proposed as part of the *treatment design* step and also already the final selection of the best option, as part of the next task *treatment validation*. Finally, the all encompassing verification of the design is still described in Section 6.1.6.

6.1.3 Platform selection

One of the initial decisions was whether to use an entire MLOps platform, individual tools, or a combination of both, namely a platform complemented by individual tools. These options are already discussed in Section 5.3. Although a decision heavily depends on the available platforms and tools, the theoretical analysis under the defined requirements suggested that using a platform-based approach supplemented by specific tools is the most suitable. This approach was, thus, chosen to be used in the design of this MLOps pipeline.

This directly led to the decision of selecting an MLOps platform. Based on the comparison of various platforms in Section 5.4, it was determined that both Azure ML and Databricks appear highly suitable for the requirements and constraints of the case study. Since no clear decision could be made based on the theoretical evaluation, both platforms were set up and tested in greater detail as part of the design process. First of all, this involved testing the deployment of the essential components of each platform via Terraform. This not only helped to uncover potential issues during the deployment of the platforms but allowed for a more thorough examination of the available functionalities within each platform. As both platforms are continually evolving, regularly adding new features or even entire components that may not have been present in the theoretical sources, this was important to ensure that the correct feature scope was considered for both platforms. Additionally, the integration and application of both platforms within the broader context of the case study environment was examined. For example, whether a platform is already used in adjacent areas within the company or is potentially promoted through an internal strategy.

The result was that the implementation effort for setting up both platforms via Terraform is comparable. Configuring a Databricks workspace seemed somewhat more complex due to its own Terraform provider for Databricks, but it offered significantly more possibilities. Through this, all configurations within the Databricks workspace can be managed directly via IaC, eliminating the need for manual or script-based configurations as often used with Azure ML. It should be noted here that the operational costs of Databricks exceed those of Azure ML because, although all resources run on Azure in both cases, Databricks incurs additional pay-per-use costs (Databricks

Units). An extended investigation within the context of the case study revealed that while both platforms are in use, Databricks is often favored, at least within the business unit. In many cases, it is used rather as a data processing platform with less focus on MLOps functionalities, but this still supports a decision in favor of Databricks. A direct examination of the deployed Databricks workspace clearly showed the data engineering platform origin of Databricks. However, it offers all the features of an MLOps platform and includes several preview features, particularly in the area of Gen AI, that were previously not identified as part of the theoretical evaluation. Likewise, the inspection of the Azure ML workspace confirmed, that it is covering all necessary MLOps components.

Ultimately, the decision was made in favor of Databricks with the final reasons as following. The complete decision and its justification can be seen in the ADR comparing MLOps platforms in GitHub[1].

- Databricks offers more options with its own Terraform provider to manage project-specific resources via IaC without manual or script based configurations.
- Databricks appears to be more in use not only within the direct case study environment but also in the broader context of the company.
- In addition to being an MLOps platform, Databricks natively provides the capability to be used as a complete data processing platform. Although this is not part of this thesis, it motivates the stakeholders in the case study environment to leverage synergies and cover multiple use cases with a single technology.
- Even though compatibility on Azure is a defined constraint of this case study, Databricks offers cloud provider independence and at least the possibility to migrate to another major cloud provider like Amazon Web Services or Google Cloud Platform in the long term.
- While Databricks is a proprietary tool, it emphasizes the use of open-source tools and provides many of its components in open-source versions. For instance, the widely used open-source tool MLflow is developed by Databricks and is provided as a core component in a managed version within Databricks[2] while Azure ML only provides MLflow compatibility in its experimentation component. Additionally, the data management platform Unity Catalog, developed by Databricks, is released as an open-source version since June 2024[3].

6.1.4 Tools selection

The decision to use Databricks as the MLOps platform provides already options for many MLOps components through the platform's built-in tools and functionalities, as shown in the platform comparison. This can also be seen in Section 5.5, where many components of the tool comparison already include functionality offered by Databricks. Therefore, for each of the MLOps component, it was assessed whether Databricks can cover it and provide a sufficient solution, or whether other tools might be better suited and warrant further investigation during the design process. This led

[1] https://github.com/gitpold/master-thesis-mlops/tree/v1.0.0/mlops-comparison/platforms
[2] https://www.databricks.com/product/managed-mlflow
[3] https://www.databricks.com/blog/open-sourcing-unity-catalog

Component	Priority	Platform or Tool
Data Labeling	Low	*open*
Data Versioning & Management	High	Databricks Unity Catalog (Delta Lake)
Exploratory Data Analysis	Low	Jupyter Notebooks in Databricks
Feature Stores	Low	Databricks Features
Code Repository	High	GitHub
CI/CD	High	GitHub Actions and Terraform
Model Training Pipeline & Workflow Orchestration	Medium	Databricks Workflows
Hyperparameter Tuning	Medium	*open*
Experiment Tracking & Metadata Store	High	Databricks Experiments (MLflow Tracking)
Model Testing & Validation	High	Giskard and Ragas
Model Registry	High	Databricks Models (MLflow Model Registry)
Model Deployment & Serving	High	Databricks Serving and LangServe
Monitoring & Observability	High	Langfuse and Databricks Inference Tables

Table 6.1: Overview of MLOps components and selected platform or tools for each component as part of the MLOps pipeline.

to many components having already one most promising tool as part of the *treatment design* step, either through Databricks or based on the theoretical evaluation. However, for some components, such as *Model Testing & Validation* and *Monitoring & Observability*, various promising options emerged. And for some of the components defined as optional, no single best option was identified. In such cases, the final decision was made as part of the *treatment validation* step, sometimes choosing the best solution after gaining additional practical insights during the implementation but sometimes even leaving the decision open to be decided individually for each use cases, allowing flexibility and adjustments to specific project requirements.

Table 6.1 presents an overview of the final selected tools for each MLOps component. In the following each of these components and their tool decisions are explained in detail, if necessary containing the findings and insights of practical comparisons as part of the *treatment validation* step for justifying the decisions. Additionally, all the decisions are documented in detail in the corresponding ADRs for each component, which are available on GitHub[4].

[4] https://github.com/gitpold/master-thesis-mlops/tree/v1.0.0/mlops-comparison/tools

Data Labeling

Since Databricks does not offer a dedicated data labeling component, the previously compared technologies are considered. Given that Azure ML is not used, employing Azure ML Data Labeling seems cumbersome. Due to the limitation that no new tools can be onboarded, the remaining options were Doccano and Labelbox. As both tools fulfill different requirements and data labeling is a low-priority component, no fixed decision was made during the MLOps pipeline design. However, the details from the comparison serve as guidance in case data labeling is required for a specific use case.

Data Versioning & Management

With Databricks as the platform, the *Data Versioning & Management* component can be handled using the corresponding functionality within Databricks. This is built on Delta Lake and offers an extensive data platform, called Unity Catalog, which has been open-sourced since June 2024.

Feature Stores

No single tool emerged as the best solution from the theoretical comparison in the previous chapter for *Feature Stores*. However, given the low priority of this component, none of the tools were further investigated or selected in the design of the MLOps pipeline. Instead, Databricks' integrated *Feature Store* component, although not previously compared, was chosen out of simplicity.

Model Training Pipeline & Workflow Orchestration

For the *Model Training Pipeline & Workflow Orchestration* component, several options were identified in the tool review. However, with Databricks in place, the use of the Databricks Workflows feature is the most logical choice. Many other options would require additional operational overhead, while the necessary functionality is already integrated and well-supported within Databricks.

Hyperparameter Tuning

The theoretical evaluation of *Hyperparameter Tuning* tools did not reveal a clear standout. Most tools provide similar features or are a bit more suitable for different libraries or use cases. Given that this component is not heavily dependent on other MLOps components and was assigned a medium priority, no fixed decision for a particular technology was made. Instead, the choice can be made depending on the individual use case, and the component can still be seamlessly integrated into the overall MLOps pipeline.

Experiment Tracking & Metadata Store

As identified in the theoretical comparison, MLflow is widely used and offers a comprehensive feature set. Since the company Databricks is also developing MLflow, it includes a fully managed instance of MLflow. Moreover, Databricks resolves the issue of missing multi-tenancy support in the standalone open-source version of MLflow by providing role-based access control. Hence, using Databricks with MLflow is an optimal solution for this component.

Model Testing & Validation

The evaluation of tools for the *Model Testing & Validation* component includes several options. Some tools were identified as not suitable for this MLOps pipeline, while others were considered potentially useful. Additionally, some tools are specialized in validating specific types of ML models or only offer broader functionality in paid versions. Therefore, using multiple tools for this component might be appropriate.

During the *treatment design* step of the MLOps pipeline, the following options are proposed: Arize Phoenix, Ragas, and Giskard for testing and validating LLMs and RAG systems. Giskard also offers functionality for testing classical ML models, and Deepchecks is another tool considered for classical ML model testing.

The setup and practical application of these tools during the *treatment validation* step, produced the following insights: Arize Phoenix is also able to cover the *Monitoring & Observability* in regard to LLM tracing. However, it does not support authentication in its free version, while the alternative Langfuse for *Monitoring & Observability* offers not only a variety of authentication options but also the ability to manage multiple projects within a single instance. For these reasons, Arize Phoenix was not utilized within the MLOps pipeline. Furthermore, testing with Giskard revealed that it is suitable for testing all types of ML models, including classical models. Deepchecks could be used the same way as Giskard by storing the generated reports in MLflow experiment runs, but requires a separate deployment of a monitoring component to display live metrics and would thus provide no additional value over Giskard. This leaves Giskard and Ragas as the decided on tools for this component.

Model Registry

The theoretical evaluation revealed few options for the *Model Registry* component. As with experiment tracking, MLflow is widely used and is directly integrated into Databricks. Additionally, the integration of MLflow with the Unity Catalog not only resolves multi-tenancy issues but also provides a comprehensive permissions model for stored models. Therefore, the integrated MLflow model registry in Databricks was chosen as the optimal solution.

Model Deployment & Serving

The *Model Deployment & Serving* component's theoretical evaluation identified several tools covering different aspects of model serving. For instance, BentoML and LangServe are no direct model serving platforms but instead facilitate the packaging of a model into a Docker container.

The use of LangServe is particularly interesting for developing chains for chatbots, especially for RAG-based systems, and was chosen as an option for hosting chains. For actual model serving, Databricks offers an integrated component that is well-suited for model serving. Given the decision to use Databricks as the MLOps platform, its model serving functionality was selected as the primary option for model deployment.

Monitoring & Observability

As with the *Model Testing & Validation* component, the theoretical evaluation of the *Monitoring & Observability* component identified several tools, some of which specialize in monitoring specific types of ML models.

For tracing of LLM calls, Langfuse and Arize Phoenix were selected as options during the *treatment design* step. Although Arize Phoenix was originally examined as part of the *Model Testing & Validation* component, it covers multiple functionalities, including LLM monitoring. For monitoring classical ML models, Evidently AI and integrated Databricks functionalities were identified as possible options. The theoretical evaluation did not reveal Databricks' monitoring capabilities; however, further testing during the design phase showed that Databricks offers simple options for monitoring deployed models, including logging all requests made and monitoring resource consumption.

During the practical validation as part of the *treatment validation* step the following insights were produced: First of all, as already described during the *Model Testing & Validation* component, Arize Phoenix was identified as having an inferior feature set to Langfuse. Thus, Langfuse was chosen for monitoring and tracing of LLM calls. Another finding was that setting up Evidently AI for monitoring the performance of classical ML models is cumbersome. At least the open-source version can only be set up as a deployed service together with an additional log forwarding component and does neither provide authentication via Azure nor multi-tenancy. Given that the testing with Databricks revealed it automatically provides general resource monitoring of model endpoints and the ability to store all model calls in an inference table for evaluation or calculation of metrics, the need for an additional model monitoring tool was eliminated.

6.1.5 Staging concept

Beyond selecting the platform and the best tools for each MLOps component, the question of how to integrate these components and establish a staging concept for the development of ML models arose. Since many of the components chosen are part of Databricks, their integration and interoperability is already ensured. The options for a staging concept for ML model development, particularly how a model is promoted between stages, is discussed in Section 5.3, where the concepts of *Deploy Model* versus *Deploy Code* were compared (see GitHub[5]). Given that the *Deploy Code* approach is recommended by Databricks, there are no impediments like very high training costs expected and this is currently the used approach within the case study environment, the *Deploy Code* concept was chosen to be applied in the design of this MLOps pipeline.

[5] https://github.com/gitpold/master-thesis-mlops/blob/v1.0.0/mlops-comparison/architecture/model-staging.png

Additionally, for the development of data products, it was decided to combine the development of the model and the application within a single repository, as this is also already the case with existing data products in the case study environment. This combines the development life cycle of the ML model and the application in one repository. However, separate development of the ML model and application code is still possible, just any changes made together on a stage are always propagated together to the next stage. Regarding the environment names, the three existing stages `DEV`, `QA`, and `PROD` used in the current DevOps concept were adopted for this MLOps pipeline.

6.1.6 Treatment Validation

The third task of the design cycle, *treatment validation*, focuses on validating the different versions of MLOps pipelines developed in the *treatment design* step. The validation of individual tools for choosing the best option has already been shown in the previous section. To achieve this the complete infrastructure of the tools was implemented to include potential issues or findings into the validation process and showcase the feasibility of the designed MLOps pipeline. The findings relating to individual tools and components were already presented with the respective components. The complete list of findings encountered during the implementation process are included in the accompanying GitHub repository[6]. The significant portion of the findings pertained to the setup of the Databricks workspace and the additional required Azure infrastructure such as networks, secret stores, and databases. However, no issues were identified that necessitate a fundamental adjustment of the MLOps platform. Rather, the individual problems highlighted that a clean deployment of the required infrastructure is essential to enable automated redeployment if necessary, and that the topic of network configuration and security should not be underestimated.

Finally, the pipeline's process and workflow are validated to ensure that all necessary MLOps components are included and that the defined constraints and requirements are met. The coverage of the components by tools can be seen in Table 6.1. Except for the components *Data Labeling* and *Hyperparameter Tuning*, all components are covered by a functionality in Databricks or additional tools. By utilizing an MLOps platform, it is also ensured that the individual components are not only compatible with each other but also well integrated, resulting in a unified MLOps pipeline. Furthermore, no tool is included that would require its own onboarding process. Rather, all relevant platforms and tools have been successfully deployed and can be used.

Regarding the fulfillment of the defined requirements, it is necessary to distinguish between the different types of requirements. While the fulfillment or non-fulfillment of constraints and functional requirements can be easily verified, the fulfillment of the non-functional requirements cannot be directly measured. Nevertheless, an assessment is provided as to whether each respective requirement is potentially fulfilled by the MLOps pipeline. The complete overview of the requirements and their fulfillment is presented in Table 6.2. It is emphasized that, except for the support of on-premise projects, which has a low priority, all defined constraint and functional requirements are fulfilled.

[6] https://github.com/gitpold/master-thesis-mlops/tree/v1.0.0/mlops-pipeline/implementation-findings

Table 6.2: List of requirements and their validation result

ID	Requirement	Validation
DC1	Compatibility with Azure	Full compatibility with Azure.
DC2	Compatibility with GitHub	Full compatibility with GitHub.
DC3	No own Kubernetes for MLOps platform	No platform or tool used needs to run on a Kubernetes platform.
FR1	Supports classical ML, Deep Learning and LLMs	While the MLOps pipeline may be more suited to some cases or sometimes need slight adaptions, the overall framework of the MLOps pipeline supports all kind of ML use cases.
FR2	Allowed at Bosch	All platforms and tools used are allowed at Bosch without further need for tool onboarding.
FR3	Integration with existing CI/CD concept	Existing DevOps as well as staging concept is adopted and extended for this MLOps pipeline.
FR4	Supports on-premise projects	Not directly possible. Could still be done via configuration of a connection between Azure and the corporate network but is not given and out of scope for this thesis.
NFR1	Maintainability	As the platform Databricks and only a limited amount of other tools are used and everything configured via Terraform, the maintenance effort is likely not low but manageable.
NFR2	User Experience	With Databricks being a paid solution and an integrated platform, it is expected to provide a good user experience.
NFR3	Reliability & Stability	With Databricks being a proprietary solution running as managed service on Azure, the expectation is for it to run reliably and stable.
NFR4	Performance & Scalability	With Databricks being a proprietary solution running as managed service on Azure as well as employing Spark, it is expected to provide good performance and scalability.
NFR5	Tool already existing at Bosch	Databricks and Langfuse have both been used already in the case study environment, even if not to the extent they are employed in this MLOps pipeline.
NFR6	Modularity & Reusability	Although Databricks itself is a proprietary platform and thus not as modular, many components it uses, like MLflow, are open source and could be used even without Databricks.

Thus, this final *treatment validation* step demonstrates that the developed MLOps pipeline includes all the necessary components and that the required platforms, tools, and additional infrastructure can be implemented. Moreover, all constraints and critical requirements for the MLOps pipeline are fulfilled, and, as far as possible within the scope of validation, it is ensured that the prerequisites for meeting the most important non-directly measurable criteria are satisfied. This validation confirms that the final MLOps pipeline is technically sound, implementable in practice, and aligned with stakeholder goals and the limitations of the case study environment. In the following section, the final developed MLOps pipeline is presented in detail, and in the next chapter, the evaluation of the pipeline for using it for the development of data products is conducted.

6.2 Final Architecture and Workflow

After the previous section detailed the development process of the MLOps pipeline using the design cycle, this section now presents the developed outcome. In Section 6.2.1, the infrastructure implemented as part of the MLOps pipeline is described, which primarily consists of Databricks and accompanying Azure resources. Subsequently, Section 6.2.2 provides a comprehensive depiction and explanation of the final MLOps pipeline. This includes an elucidation of which platforms and tools are employed for each MLOps component, how they interact within the pipeline, and a presentation of the staging concept. Finally, in Section 6.2.3, a template repository is introduced, intended as a starting point for the development of data products. This template already contains the foundational structure for applying the MLOps pipeline, along with a user guide for utilizing both the template and the MLOps pipeline for the development of data products.

6.2.1 Infrastructure

In this section, the infrastructure deployed for the MLOps pipeline is presented. The entire infrastructure is managed and deployed following the IaC principle using Terraform. The components consist of the established platforms and tools that have deployable components and are not merely libraries for local application. Specifically, this includes the platform Databricks, referred to hereafter often as a Databricks workspace, and the tool Langfuse. Additionally, supporting Azure components such as networking, secret stores, and databases are required, which are also part of the MLOps infrastructure. Finally, the deployable components of a data product to be developed are also shown as they require access to the rest of the MLOps infrastructure. Figure 6.1 therefore presents a comprehensive architectural diagram of the deployed infrastructure, including the deployed components for an example data product. It is apparent that this architectural diagram includes only one instance each of a Databricks workspace and Langfuse. As defined in the design process and described subsequently in Section 6.2.2, the staging concept of the MLOps pipeline comprises three stages: `DEV`, `QA`, and `PROD`. Although these could theoretically be mapped onto a single infrastructure instance, the overall design envisions three separate Databricks workspaces (one for each stage). Due to time constraints, only the `DEV` instance can be implemented within the scope of this thesis. However, since the other instances would be identical, the infrastructure can be presented and explained here using the example of the `DEV` infrastructure.

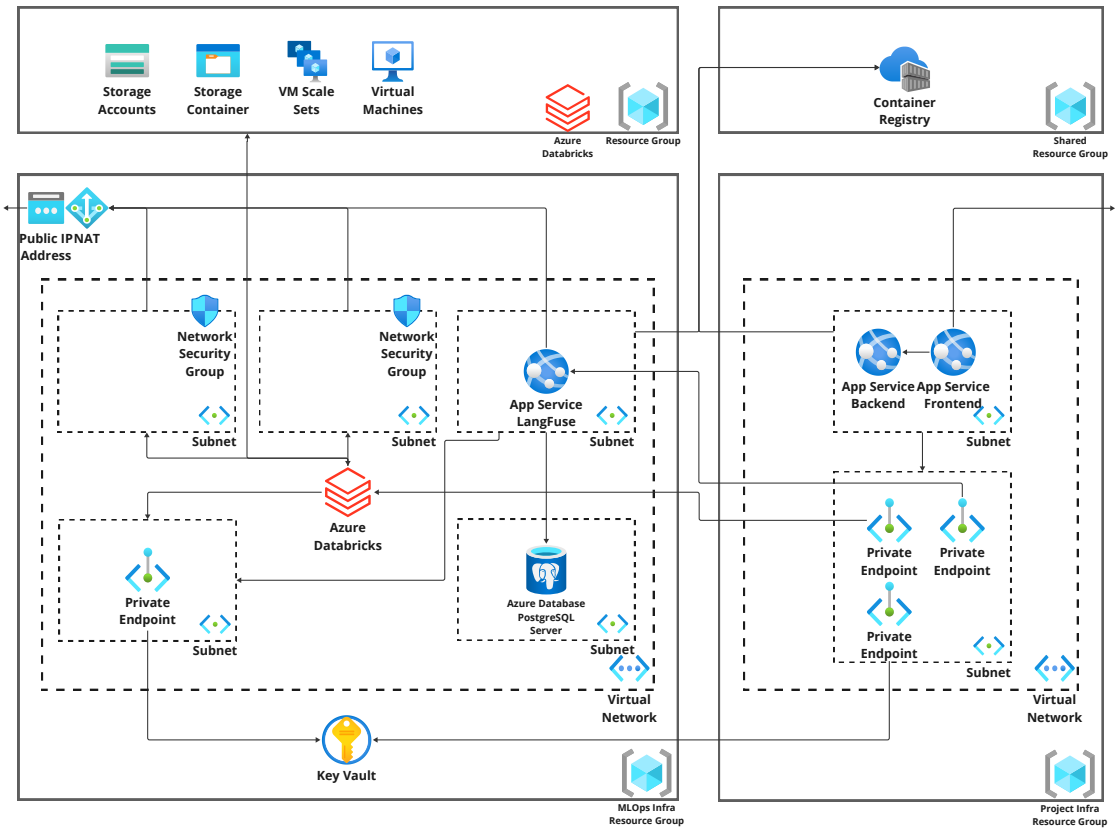


Figure 6.1: Architecture overview of MLOps infrastructure components together with infrastructure for a sample project.

The architecture overview presented in Figure 6.1 is divided into four sections, each corresponding to a resource group within Azure (a container for holding resources). The following describes the infrastructure, organized according to these resource groups:

Bottom left: This resource group contains the main components of the MLOps infrastructure and includes the resources deployed during this master's thesis via a TERRAFORM stack. It encompasses a virtual network with multiple subnets. Within this resource group, a key vault is included for storing secrets related to the DATABRICKS and LANGFUSE deployments, made available via a private endpoint within a subnet. Another subnet hosts a PostgreSQL database used by LANGFUSE for data storage. LANGFUSE itself is deployed in a separate subnet as an Azure Web App. It accesses both the key vault and the database and is accessible via a public domain, although access is restricted to the Bosch Internet Protocol (IP) range. Finally, the DATABRICKS workspace is included in this resource group. It also accesses the key vault and contains two necessary subnets managed by the service itself. Additionally, the resource group contains a Network Address Translation (NAT) Gateway with a fixed IP address, which can be used to implement IP whitelisting at consuming services.

Top left: In addition to the two managed subnets, the DATABRICKS workspace creates a managed resource group containing Azure resources such as storage accounts and virtual machines. These resources are dynamically created by DATABRICKS when users store data or create compute resources within DATABRICKS.

Bottom right: This resource group represents the deployed components of a sample data product project. It again contains a virtual network for organizing network access, with two subnets in this case. Similar to the Langfuse deployment, one subnet hosts both the backend and frontend of the data product, each deployed as an Azure Web App and exposed via a domain, but accessible only within the Bosch IP range. To enable these applications to communicate with the key vault, the Langfuse instance, and most importantly the Databricks workspace serving the ML model, the secure connections to these resources are integrated via private endpoints into another subnet within the virtual network.

Top right: This resource group already existed as part of the case study environment. It is intended for resources shared between projects within the department. Although it contains more resources than depicted here, only the container registry is utilized in this context, serving as the registry for storing and pulling Docker images of Langfuse and the applications themselves.

Since some tools were already in use or necessary resources were available within the context of the case study, existing examples could be leveraged during implementation. For instance, there is an existing example of a Terraform deployment for Langfuse and a Container Registry that can be used directly. However, because one of the intended uses of this solution is to potentially provide this MLOps pipeline as a blueprint for other areas, dependencies on existing systems or infrastructure should be minimized wherever possible.

Therefore, the deployment for Langfuse was adapted based on the existing example and integrated into the repository for the MLOps infrastructure. For storing Docker images, however, the existing Azure Container Registry is utilized. This choice is justified by the fact that it saves both time and costs and provides a more uniform and easily understandable environment for developers when only a single Container Registry is in use. This rationale applies not only to the immediate department but also when this blueprint is used in other areas. Since a Container Registry is a fundamental component, it can be assumed that one already exists in most cases, potentially even outside of Azure, such as when GitHub Packages is used. Therefore, it is sensible to use the existing Container Registry already in use, and if one is not available, it can be incorporated into the infrastructure blueprint with minimal effort.

Another utilization of existing functionality pertains to the template repository for data products, which is defined in GitHub and described in more detail in Section 6.2.3. The case study department employs an automated process that not only enables cloning the template repository for a new project but also configures a significant portion of the required secrets and variables in the GitHub repository. Additionally, this process automatically creates a new Resource Group in Azure for the project, along with the basic Terraform configuration for the new project. Since this automation not only simplifies the project setup considerably but is also the only supported method for creating new projects within the context of the case study, it is applied to the MLOps example projects as well. Although it cannot be assumed that a similar process is always available, the target state achieved by this automation can also be reached through manual steps. Furthermore, when using Azure and GitHub, along with GitHub Actions and Terraform for deployment, this process can be considered a given prerequisite.

Databricks Configuration

What is not apparent from the architectural diagram in Figure 6.1 is that not only the Azure Databricks workspace is deployed via Terraform. Using the Terraform provider for Databricks, the Databricks workspace and its internal resources are also configured as part of the MLOps infrastructure setup. Upon the initial deployment of the Databricks workspace, the following components are configured:

- Access limitations to the Azure Databricks workspace based on an IP whitelist, allowing only the Bosch IP range in order to fulfill necessary security requirements.
- A secret scope backed by an Azure key vault for storing secrets that are later accessible in Databricks.
- A Git credential object for pulling and pushing changes from Databricks to a repository in GitHub.
- The creation of a Personal Access Token (PAT) for Databricks to be used in the pipeline of the data product projects.
- A directory inside the Databricks workspace for the respective stage.
- A custom cluster policy for predefining the compute resources users are allowed to create inside Databricks.

Having this included as IaC inside of a repository, provides the possibility to easily spin up the needed MLOps infrastructure in other locations as well without the need for lengthy manual configurations.

6.2.2 Pipeline

After presenting the implemented infrastructure components in Section 6.2.1, this section now introduces the developed MLOps pipeline itself. To provide a comprehensive overview, Figure 6.2 illustrates the MLOps pipeline, including the individual components as well as the connections between them. Additionally, all three stages are depicted. For a clearer visualization of the components and processes within a single stage, Figure 6.3 offers a view focused solely on the `DEV` stage. Both representations of the MLOps pipeline are loosely based on the depiction of the MLOps concept by Databricks in their e-book *The Big Book of MLOps* [BKTT22].

The relevant points regarding the staging concept are that it is based on the existing staging concept from the DevOps process and sees the model development as part of the data product. This means that the application on the respective stage always uses the corresponding version of the ML model and not always a productive version of an ML model developed in a separate lifecycle. As the existing staging concept from the DevOps process already includes a consistent deployment of all components on the `DEV` stage, this is also extended to all MLOps components (otherwise, for example, only model training without validation or deployment could take place on `DEV`). A major advantage of adopting the existing staging concept is that the three stages `DEV`, `QA`, and `PROD` continue to represent the corresponding stage from the end user's perspective and it can be developed on `DEV`, user tests can be carried out on `QA`, and a stable version can be used in production on `PROD`. It

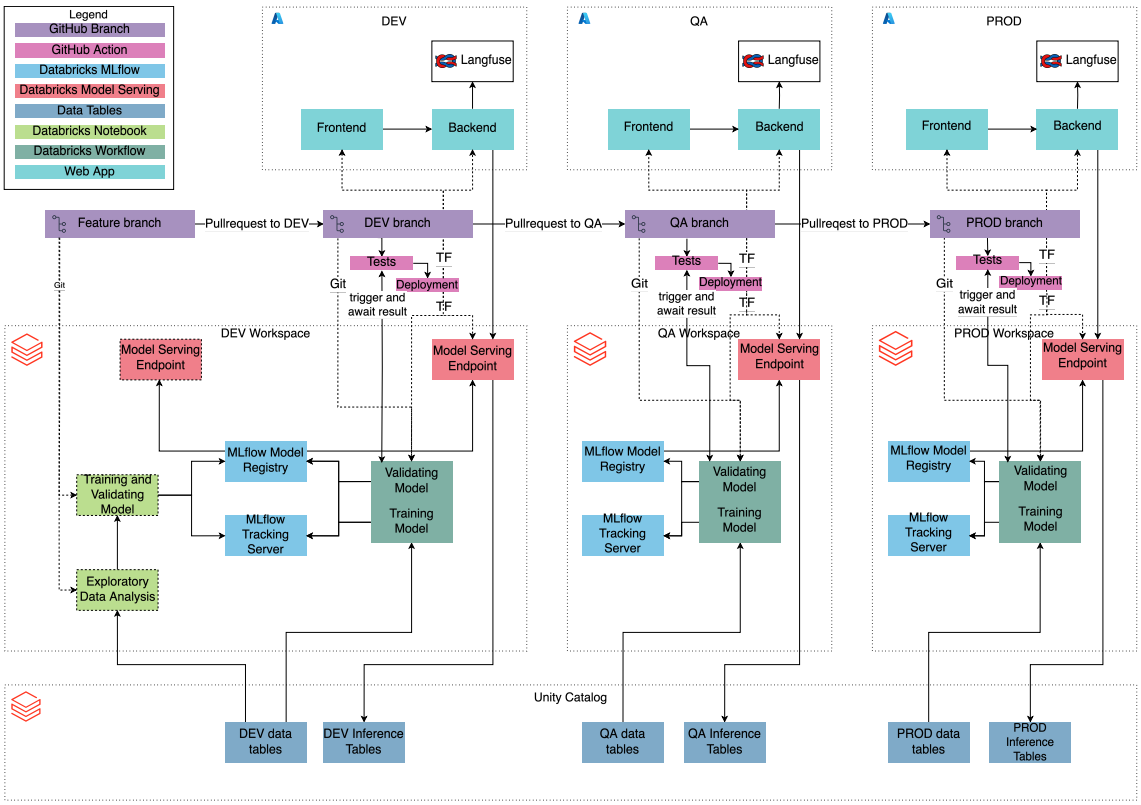


Figure 6.2: Overview of MLOps pipeline with view on all stages.

also simplifies the management and testing of the stages if there are no differences between the stages. As mentioned in Section 6.2.1 the stages `QA` and `PROD` are not implemented in practice in the scope of this thesis. However, as the `DEV` stage already includes all necessary components and must just be duplicated on the respective `QA` and `PROD` branches, this should be possible without further problems.

From the perspective of the defined components, the MLOps pipeline is covered as follows:

***Data Labeling*:** The *Data Labeling* component is not considered in this MLOps pipeline. In case it would get included, it would take place apart from the whole depicted process and result in labeled data which would get stored in the data tables (dark blue) in the Unity Catalog.

***Data Versioning & Management*:** This component is covered by the Unity Catalog, depicted on the bottom, containing data tables (dark blue) for storing the data needed for model development.

***Exploratory Data Analysis*:** This component, covered by Jupyter Notebooks in Databricks (light green), is used for exploring the data available in the Unity Catalog via a notebook on the users own feature branch checked out in Databricks (or locally). This may even include an initial experimentation with model training for fast prototype development. Both notebooks depicted in the MLOps pipeline have a dashed frame indicating that these steps are not automated through the process but depend on the developer manually doing them.

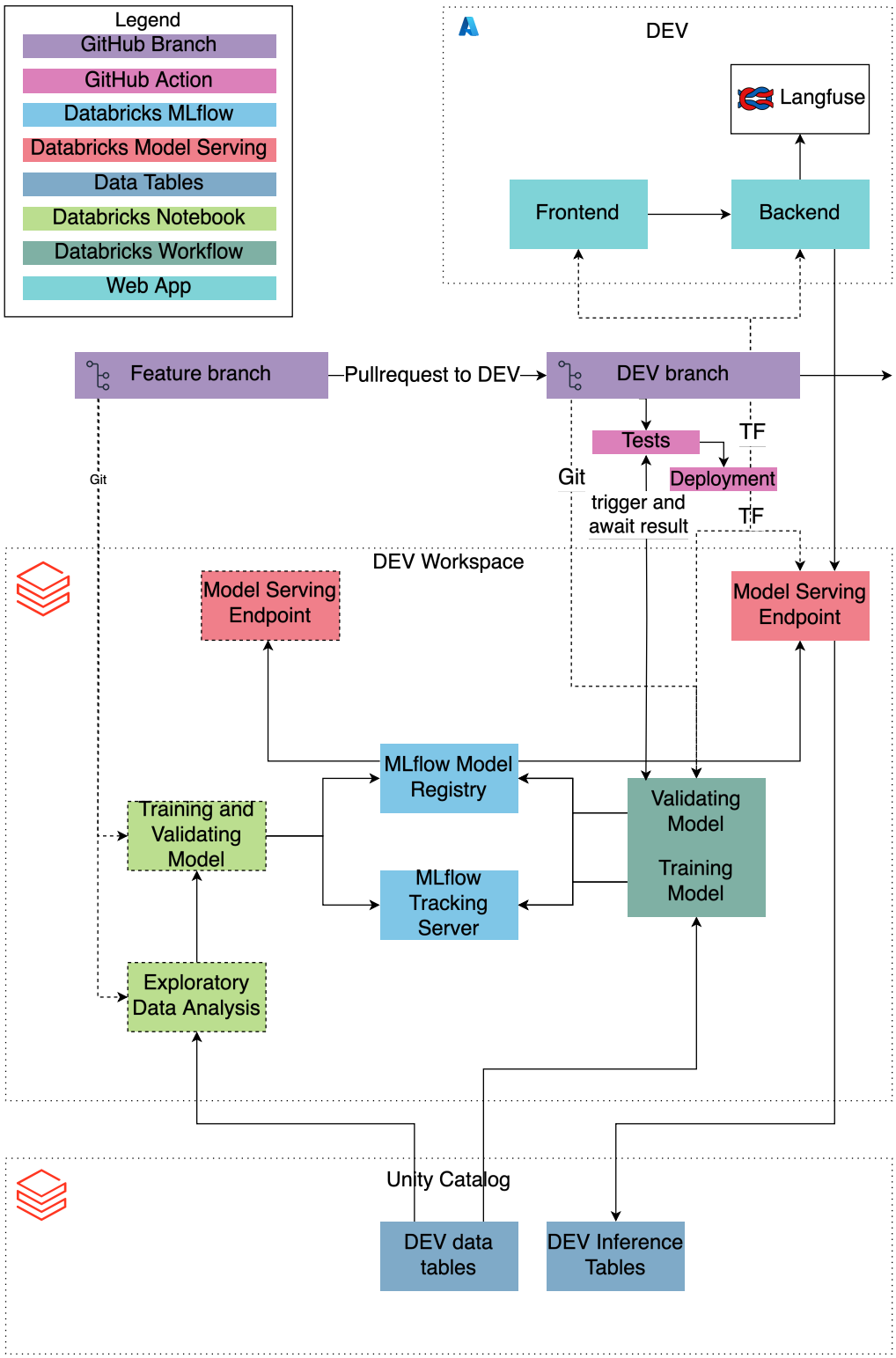


Figure 6.3: Overview of MLOps pipeline with view on `DEV` stage.

***Feature Stores*:** The *Feature Store* component is not considered in this MLOps pipeline but it would leverage the Unity Catalog to serve features, similar to data tables, to be used in model development.

***Code Repository*:** This component is covered by GitHub. The view depicts different branches (purple) of the repository and shows how these map to the different stages. Not only the application and model training code, but also the infrastructure deployment and CI pipeline definitions are stored inside of the Git repository.

***CI/CD*:** GitHub Actions and Terraform are used for providing the automation of the MLOps pipeline. The GitHub Actions pipelines (pink) are used to trigger the training and validation of a model as well as the building of the Docker images for the applications. These are configured to run on push, or on pull request, to a branch of the respective stage. Afterwards the Terraform deployment job deploys the latest version of the model to the model serving endpoint (red) on Databricks as well as the backend and frontend (turquoise) to Azure Web Apps for the respective stage.

***Model Training Pipeline & Workflow Orchestration*:** Covered by the jobs and workflow feature in Databricks, this component is used to run the model training and validation code (dark green) on the latest data. These workflows are automatically triggered by the CI pipeline.

***Hyperparameter Tuning*:** This component is not directly considered and not depicted in this MLOps pipeline. But as this includes normally the use of a library during the training of the model, it would take place as part of the model training workflow (dark green).

***Experiment Tracking & Metadata Store*:** For tracking of the experiments during model development the MLflow Tracking server (light blue) integrated into Databricks is used. Every time a model is trained and validated the run with all its parameters and metrics is stored in an experiment in MLflow. This includes the automated model training via workflow triggered by the CI pipeline, but also the experimental development on a feature branch from a notebook.

***Model Testing & Validation*:** Here, too, only libraries and no standalone component are used. This means the chosen Giskard and Ragas are applied during the model validation. This takes place during the automated model validation workflow (dark green) but can also be applied already during the initial experimentation via a notebook (light green).

***Model Registry*:** For storing the generated models the MLflow Model Registry (light blue) integrated into Databricks is used. Again, both the automatically and the manually built models are getting registered in the model registry.

***Model Deployment & Serving*:** For the *Model Deployment & Serving* component the Databricks Model Serving Endpoint and LangServe are used. The model serving endpoint (red) in Databricks is used to deploy the registered models from the model registry and make them accessible via API. In the case of the feature stage this component is displayed with a dashed frame indicating that this is not automatically created as part of the automation of this MLOps pipeline. Rather, this can be done manually by the developer in case a new model developed on a feature branch needs to get directly tested, for example, in the corresponding local application version of this branch. The model serving endpoints without the dashed frame are getting automatically updated to the latest model version for each stage. Additionally, for chatbots with RAG use cases, the developed

chains can directly be defined in the backend application (turquoise) using the LangServe library. In this case the backend still makes a call to the model serving endpoint in Databricks, however, it does not call the developed chain but rather only a LLM also available there via API.

***Monitoring & Observability*:** The tool Langfuse (white) used for tracing of LLM responses is available as a separate component hosted inside of Azure. It is used only for this type of ML use cases and is configured directly in the code of the backend application (turquoise). The integrated Databricks monitoring logs all the requests to a model endpoint with request and responses and saves them to so-called inference tables in the Unity Catalog enabling for example later performance evaluations.

6.2.3 Template and Usage Guide

After presenting the architectural overview of the deployed MLOps infrastructure in Section 6.2.1 and the designed MLOps pipeline in Section 6.2.2, all the necessary elements are in place to apply this MLOps pipeline for the development of data products. However, even though all required components and the conceptual framework are available, utilizing the pipeline is not trivial. While the MLOps pipeline concept can certainly be applied to new projects using the corresponding tools and implementing the development process and deployment accordingly, the setup and application of the MLOps pipeline still involves effort and requires appropriate prior knowledge. This is not desired especially when the goal is to simplify and accelerate data product development.

To make the application of the MLOps pipeline during development as straightforward as possible, a template repository was created that contains the basic structure for using the MLOps pipeline. This includes predefined CI pipelines and IaC setups, as well as example code for utilizing individual MLOps components like the MLflow Tracking Server or Langfuse. In addition to the general documentation of the MLOps pipeline, as provided in this thesis, a targeted user guide was developed. This guide describes the intended application of the MLOps pipeline for data product development in concise and clear steps.

The user guide offers a detailed setup for initiating a new project based on this template, an overview of the respective MLOps components, and a step-by-step guide for developing the ML model and the entire data product using the MLOps pipeline. While this documentation is not intended to be a comprehensive guide to all the concepts and tools used, it aims to provide a sufficient starting point so that developers without extensive prior experience in the MLOps domain can apply this pipeline and develop a data product.

Template Repository

To provide a simple starting point and enable rapid development of a data product, a template repository was provided. This repository already contains fundamental components, but primarily the necessary Terraform configurations and predefined GitHub Actions pipelines. It also includes examples for using specific tools, as well as a brief README explaining the contents of the template. In practice, it became apparent that the contents of the template can vary slightly depending on the type of ML project. For this reason, two slightly different templates were created

as part of this thesis. One follows the use case of a classical ML model, and the other was designed for use cases involving a LLM with a RAG approach. In Section 7.1, two sample use cases for these templates are presented.

Despite these minor differences, both templates are described together in this section. They include the following components, each organized into separate directories:

- **Frontend**: Contains code for a minimalist Streamlit app, allowing users to interact with the data product.
- **Backend**: A minimal FastAPI backend. This application calls a model endpoint in Databricks and provides an API for the frontend. In the case of the RAG use case, this backend additionally includes the LangServe library and an example for defining a LangChain, which accesses a foundation model and a vector search endpoint in Databricks.
- **Model**: This directory contains an example notebook with relevant snippets for data exploration and initial model prototyping, as well as example code for working with MLflow. Additionally, placeholders for `train.py` and `validate.py` are included, to be executed within a workflow as part of the MLOps pipeline.
- **Deployment**: Contains the configuration for a prepared Terraform stack. This stack includes deploying the necessary Azure resources, such as the web apps for the backend and frontend. Furthermore, as described in Section 6.2.1 during the setup of the MLOps infrastructure, the Terraform provider for Databricks is used here. This allows all initially required resources and additional components within Databricks for this project to be configured via IaC. The resources may vary slightly depending on the use case and template used, but generally, the template enables the provisioning of the following resources:
  - A repository inside Databricks for the respective project within the environment directory.
  - An experiment for the project in the MLflow Tracking server.
  - A model for the project in the MLflow Model Registry.
  - A workflow in Databricks for executing tasks related to training and validating models.
  - Optionally, a model serving endpoint in Databricks.
  - Optionally, a vector search endpoint in Databricks.
- **CI pipeline**: The definition of the GitHub Actions pipelines. These pipelines encompass the building and deployment of Docker containers for the frontend and backend, as established in the DevOps concept. Additionally, they incorporate the automation for triggering the model training workflow and deploying a model endpoint at each stage, as presented in Section 6.2.2.

By using this template, all initially required components are configured and deployed, and the workflow for the MLOps pipeline is established. This allows developers to focus directly on the development of the ML model and the application for the data product.

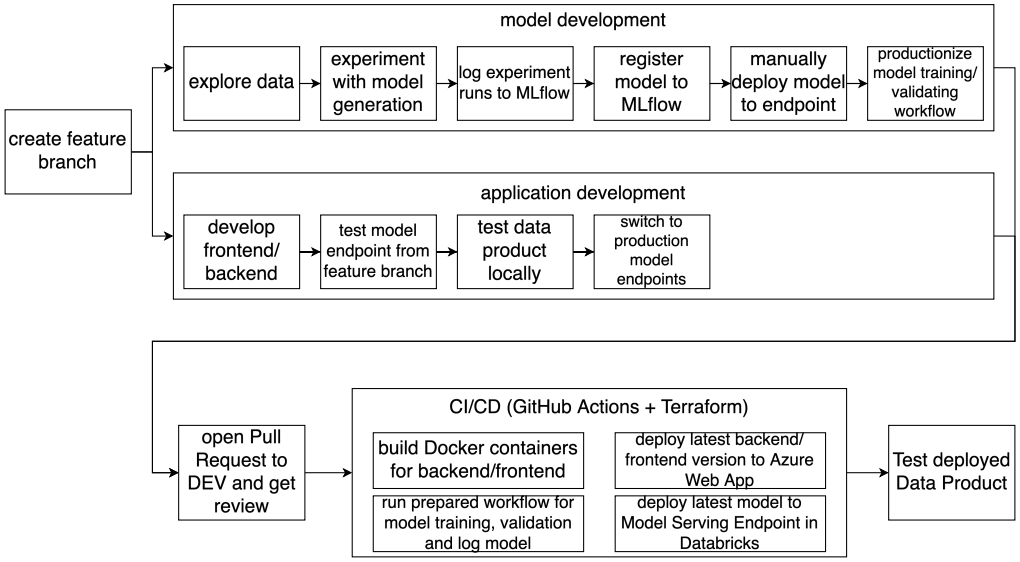


Figure 6.4: Development flow for working with the template.

User Guide

In addition to the general documentation of the MLOps pipeline and the template described above, a brief user guide was created. This guide focuses primarily on setting up a new project based on the template and outlining the development workflow. The complete steps can be found in the appendix under Appendix E. The initial setup includes steps for creating a new repository from the template, the initial configuration of the Databricks workspace for the developer, and how to locally run the application.

Subsequently, the workflow for a developer is described, detailing how to use the MLOps pipeline to develop a first version of the data product and deploy it to the `DEV` stage. This workflow is also illustrated in Figure 6.4. At a high level, this process follows the Git flow: after creating a new feature branch, development and testing are conducted on this branch, followed by opening a pull request to the `DEV` branch. Upon review and approval, the CI pipeline automatically executes all steps up to deploying the data product on the `DEV` stage, which can then be tested and used by the `DEV` version of the application. The only new aspect for developers is that, in addition to traditional application development, the development of the model also occurs on the feature branch, and the training and deployment of the model are integrated into the automated CI/CD pipeline.

6.2.4 Summary

Thus, in this phase, an overall fully functional MLOps pipeline was developed. The complete necessary infrastructure was set up, and a template including documentation for direct usage of the MLOps pipeline was prepared. This allows, in the following Chapter 7, an application of the MLOps pipeline on sample use cases in order to evaluate the effectiveness of the pipeline in a practical setup.

7 Evaluation of the Data Product Development Pipeline

After having evaluated various MLOps approaches and, based on these insights, having designed and implemented an MLOps pipeline, including setting up the infrastructure, the question of the pipelines effectiveness arises. To address research question **RQ3** and the third objective of this thesis (Goal 3), the MLOps pipeline was tested in a hackathon-like setup with test users implementing sample use cases. For this purpose, the methods described in Section 4.3 were applied to gather insights and feedback during the user tests. This involved observing the participants during the hackathon, conducting a final feedback session based on the focus group methodology, and administering a short anonymous survey that each participant completed. Since the MLOps workflow can differ depending on the type of ML project (classical ML model or using an LLM, for example in a RAG approach) and there are two test groups, the two different use cases described in Section 4.3.1 were tested.

Section 7.1 describes the created reference implementations for the chatbot use case described in Section 4.3.1 and the computer vision use case shown in Section 4.3.1. The findings from conducting the hackathons to test the MLOps pipeline are presented in Section 7.2. This includes an overview of interesting observations during the execution, as well as documentation of all interventions by the observer of the MLOps pipeline. Furthermore, the results from the concluding focus group interviews and the outcomes of the anonymized participant surveys are presented. In Section 7.3, the collected feedback is evaluated and analyzed in relation to the research question **RQ3** to assess the effectiveness of the designed data product development pipeline. Potential differences between the two test groups are also highlighted. Additionally, it is concluded which steps and improvements would be still needed to apply the data product development pipeline to a new, real-world use case.

7.1 Sample Use Cases

The implementation and final deployed infrastructure of MLOps platforms and tools is already described in Section 6.2.1 and the MLOps pipeline and its workflow are explained in Section 6.2.2. Furthermore, a template repository exists, containing custom Terraform components, prepared GitHub Actions for automation, and a README explaining how to use this MLOps pipeline, as detailed in Section 6.2.3. To test the effectiveness of the designed pipeline, it must be evaluated on realistic use cases. For this purpose, the use cases described in Section 4.3.1 were used for testing the MLOps pipeline in a hackathon-like user test. Having two different ML use cases is advantageous as this allows to verify the MLOps pipeline with different types of ML projects.

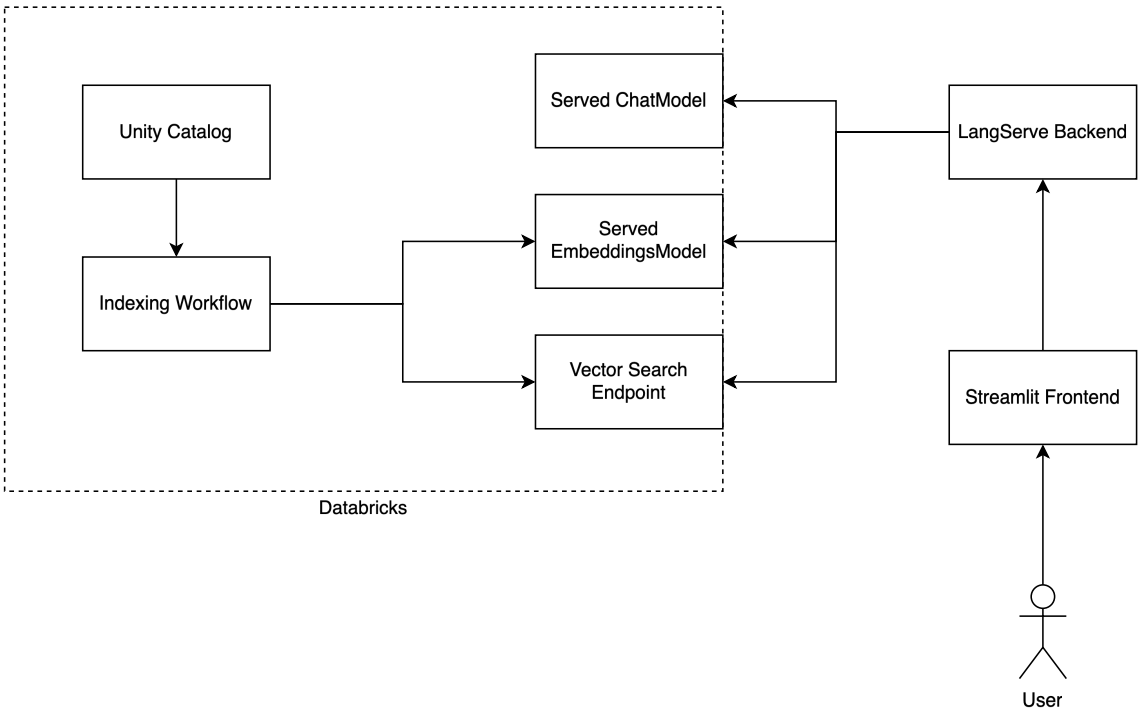


Figure 7.1: High-level architecture overview for sample use case 1.

In order to verify that both use cases were suited, they were first implemented as reference implementations. Section 7.1.1 describes the reference implementation of the chatbot use case and in Section 7.1.2 the reference implementation of the classification use case is presented. The insights from these test implementations are used for the custom instruction guides provided to the test user groups. They build on top of the general user guide presented in Section 6.2.3 and incorporate additionally relevant, use case specific steps.

7.1.1 Sample Use Case 1: Chatbot answering questions on a climate report

As outlined in the requirements, the designed MLOps pipeline aims to support all types of ML projects. For covering the LLMOps domain, the chatbot use case for answering questions about a climate report as described in Section 4.3.1 is applied. The following describes the architecture and application of the MLOps pipeline to a reference implementation of this use case.

Figure 7.1 presents the high-level architecture of this sample use case. The climate report used for the retriever component is stored in the Unity Catalog. A workflow in Databricks processes this report by chunking it and storing the text with embeddings in a vector store within Databricks. For embedding generation and text generation, models available in Databricks are utilized. While implementing this use case could involve any vector store and models for embedding and text generation via APIs from different providers, this sample project demonstrates the use of components directly within Databricks. Thus, for embeddings and chat functionalities, models are used that, although not hosted within the Databricks workspace, are made available directly in Databricks via the serving component. Specifically, these are `GTE Large (En)` and `Meta Llama 3.1 70B Instruct`.

In the final architecture, although the development of the RAG chain can be tested from a notebook in Databricks, the chain itself is not provided via the Databricks Serving components in this version. For fast testing of the RAG chain, however, this would be possible. Instead, the backend is implemented using FastAPI with LangServe, which contains the actual RAG chain, implemented via the framework LangChain, and accesses the LLM and vector store endpoint in Databricks accordingly. To facilitate easy interaction with the chatbot for users, the frontend consists of a Streamlit app featuring a simple chat interface. Hosting the RAG chain via LangServe allows the frontend to directly connect to the runnable chain, enabling, for instance, to stream the responses from the chain. A screenshot showing the frontend of the reference implementation for this data product is provided in the Appendix in Figure F.2.

As this use case is not a classical ML use case that trains a model but instead uses an existing LLM, this shows how the designed MLOps pipeline can be adapted and still applied. While the overall workflow follows the MLOps pipeline already depicted in Figure 6.3, the appendix contains a custom overview of the complete MLOps workflow in Databricks for this use case depicted in Figure F.1. For this reference implementation only the `DEV` workspace and staging up to the `DEV` stage were implemented. Feature branches are employed for initial data exploration and testing the development of a RAG chain. This process includes creating a vector search endpoint and populating it with test data. The chain then accesses the vector search endpoint and the LLM available with Databricks. The model runs are stored in the MLflow experiment tracking component, along with complete traces of the chain. For rapid prototyping of the application, the model can also be manually pushed from the feature branch into the MLflow Model Registry and served as an endpoint via the Databricks Serving component. The actual automated part of the MLOps pipeline occurs again with a push to the `DEV` branch. No custom model is trained here; instead, the corresponding chain is built and validated, for example, based on performance on a test set or on security metrics such as checking for hallucination. These workflows for validating the model and an additional workflow for automated indexing are triggered via GitHub Actions pipelines and are fully managed in Terraform. The option to automatically serve the chain via the Databricks serving endpoint is shown grayed out in the diagram. As previously mentioned, this is not implemented in this use case since the chain itself is implemented in the backend with LangServe to allow not only access via API but to access the chatbot response in a streaming manner via LangServe. Furthermore, monitoring and tracing of the chatbot calls are not depicted here, as they are not implemented in Databricks but directly in the backend, which sends all traces to a Langfuse instance.

7.1.2 Sample Use Case 2: Classification of handwritten digits

To illustrate the application of the MLOps pipeline with a classical ML project, the classification use case as described in Section 4.3.1 is applied. It trains a model on the MNIST dataset for classifying handwritten digits. The following describes the architecture and application of the MLOps pipeline to a reference implementation of this use case.

Figure 7.2 provides a high-level overview of the architecture of the developed data product for this use case. The MNIST data used for training the model is stored in the Unity Catalog in Databricks. A workflow in Databricks trains and validates the model and stores the trained model in the model registry within Databricks. From there, the latest model is hosted in the Databricks Serving component and made available for use via an API. Additionally, a simple backend and frontend is included, in this case, FastAPI for the backend and Streamlit for the frontend. The backend interacts

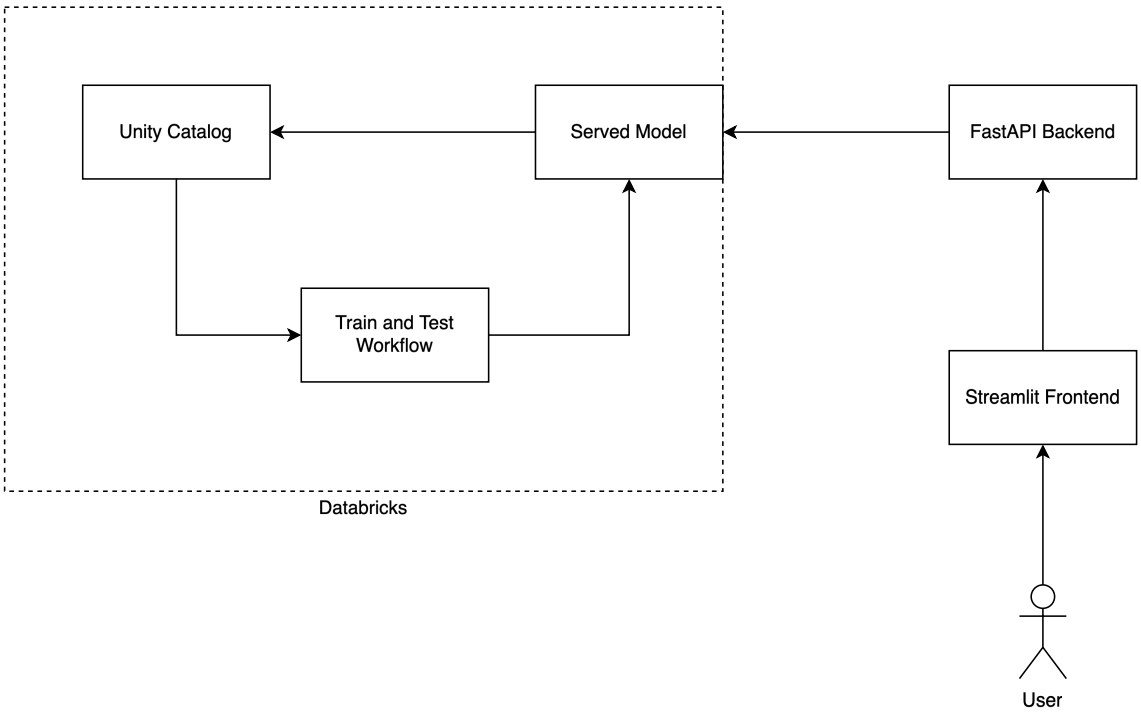


Figure 7.2: High-level architecture overview for sample use case 2.

with the model endpoint on Databricks and performs transformations on the model input and output. The frontend allows users to draw digits and displays the classification results. The arrow from model serving to the Unity Catalog represents the logging and storage of model calls for future analysis. A screenshot showing the use of the data product via the frontend is presented in the appendix in Figure F.4.

Up to this point, the architecture describes a data product that utilizes an ML model but does not delve into the development process and the application of the MLOps pipeline. While the overall workflow follows the MLOps pipeline already depicted in Figure 6.3, the appendix contains a tailored overview of the complete MLOps workflow in Databricks for this use case depicted in Figure F.3. As for the first sample use case, only the `DEV` stage were implemented. The components depicted with dashed lines represent manual components, such as the initial data analysis or testing of model development on a feature branch, up to the manual deployment of the model in Databricks for quick testing from outside Databricks, such as from a local backend. The components shown with solid lines correspond to the automated steps within the MLOps pipeline that are integrated into the CI/CD pipeline and are executed automatically upon pushing to the `DEV` branch. In this case, this includes training and validating the model as well as the automated deployment of the model. The model deployment, as well as all other resources and workflows within Databricks, are managed as IaC via Terraform.

Role	Dev	Ops	CI/CD	ML
Data Engineer	Intermediate	Beginner / Intermediate	Beginner / Intermediate	Beginner / Intermediate
Data Engineer	Intermediate / Expert	Beginner / Intermediate	Beginner / Intermediate	Intermediate
Data Scientist / ML Engineer	Intermediate / Expert	Beginner / Intermediate	Beginner / Intermediate	Expert
Full Stack Developer / Cloud Engineer	Expert	Intermediate	Intermediate	Beginner / Intermediate

Table 7.1: Roles and experience levels of test user group 1.

7.2 Evaluation Results

This section presents the results of the user tests conducted to evaluate the developed MLOps pipeline. Specifically, it relates to testing the MLOps pipeline in a hackathon-like setting for implementing the sample use cases described in Section 4.3.1. As previously noted, testing the deployment of the necessary infrastructure, including the DATABRICKS platform, additional tools, and supporting components in Azure, is not included, as this is not feasible within the time constraints of the user tests. Since the MLOps pipeline is tested with two different user groups, each working on different sample use cases, Section 7.2.1 will separately detail each user group. Then, the observations made during the execution of each hackathon and the outcomes achieved by each group are presented in Section 7.2.2. Subsequently in Section 7.2.3, the feedback obtained from both user tests is summarized, categorized into positive and negative responses, along with suggestions for improvements. Finally, the results of the anonymous survey are presented in Section 7.2.4.

7.2.1 Test User Groups

The first test user group consisted of developers directly from the department of the case study for which this MLOps pipeline has been developed. Their requirements, experiences, and existing processes have directly influenced the outcome of the MLOps pipeline. In this user group four participants took part in the hackathon. Their job roles and self-assessed experience levels in the fields of development, operations, CI/CD, and ML are described in Table 7.1. Notably is that the focus of this test group is on data engineers and data scientists. This means they have extensive experience in development and ML, while comparatively less experience with operations and CI/CD exists within the group. This test group conducted the hackathon by testing the MLOps pipeline with the RAG use case described in Section 7.1.1. Important to notice is, that this hackathon has been performed fully remote. Collaboration was given by having all participants together inside of one Microsoft Teams call.

Role	Dev	Ops	CI/CD	ML
Platform Engineer	Expert	Intermediate / Expert	Intermediate / Expert	Beginner / Intermediate
Data Scientist & SW Engineer	Intermediate / Expert	Beginner / Intermediate	Beginner / Intermediate	Intermediate / Expert
Cloud Network Developer	Beginner / Intermediate	Intermediate / Expert	Intermediate	Beginner
Cloud Solution Architect	Beginner / Intermediate	Expert	Intermediate / Expert	Beginner
Data Engineer & Scientist	Intermediate / Expert	Beginner / Intermediate	Beginner / Intermediate	Intermediate / Expert

Table 7.2: Roles and experience levels of test user group 2.

The second test user group consisted of five participants. They are from the same company but have no direct connection to the department of the case study environment. The participants are familiar with Azure and GitHub, as these technologies are widely used within the company. However, they are not familiar with the specific requirements or existing DevOps process from the case study environment that have influenced the MLOps pipeline. Table 7.2 shows the job roles and self-assessed experience levels of each participant. It is noteworthy that this test group is more homogeneously distributed in terms of job roles. For all four of the queried competency areas (development, operations, CI/CD, and ML), there are participants with at least an intermediate to expert level of expertise in the field. This test group conducted the hackathon by testing the MLOps pipeline using the use case for classifying handwritten digits with the MNIST dataset, as described in Section 7.1.2. In comparison to the first test user group, the hackathon with the second test user group was performed with all participants on site together in a meeting room.

7.2.2 Hackathon Observations, Involvements and Result

For the execution of the hackathons, each test user group received a set of instructions. These were based on the user guide described in Section 6.2.3 and included detailed descriptions of the initial setup steps. Additionally, they contained step-by-step instructions defining sub-goals to ensure comprehensive testing of the entire MLOps pipeline during the development of the data product. During the hackathons, the author of this thesis was present to act as an observer. Although a purely passive observation would be preferable for evaluation purposes, the observer, as described in Section 4.3.2, was also available to answer questions or intervene proactively if necessary. During the execution, it became apparent that such involvement was necessary to achieve a testing of the entire MLOps pipeline on the sample use cases within the limited time frame.

Type of Problem	Only Observed	Given Hint	Answered Question	Total Amount
Documentation	3	1	0	4
Tool	8	10	4	22
Process	3	0	3	6

Table 7.3: Amount of encountered issues for test group 1 categorized by type of problem and separated by observation and involvement.

Type of Problem	Only Observed	Given Hint	Answered Question	Total Amount
Documentation	3	1	0	4
Tool	7	6	2	15
Process	6	3	4	13

Table 7.4: Amount of encountered issues for test group 2 categorized by type of problem and separated by observation and involvement.

The full protocols of observations and observer involvements for both hackathons can be found on GitHub[1]. An overview of the number of encountered issues, categorized by the type of problem, is provided here. The observer's involvements are separated into hints given by the observer to support the test user group and direct questions asked by the participants. While participant questions are prompted by their awareness of problems or uncertainties, the hints are mainly given to prevent unnecessary mistakes or to confirm assumptions, ultimately aiming to expedite progress to cover as much of the MLOps pipeline as possible within the limited time frame. The types of problems that have been recorded in the protocols are:

- **Documentation:** The documentation was wrong or essential information is missing.
- **Tool:** The participants had trouble using a specific platform, tool or library.
- **Process:** The workflow of the MLOps pipeline itself led to problems.

Table 7.3 shows the number of encountered issues for the first test user group, and Table 7.4 provides the same for the second group. In each hackathon, the total number of issues identified was 32. While the overall distribution between issue types is similar, the first user group encountered more tool-related issues than the second group, whereas the second test user group faced more process-related issues than the first group. Overall, tool-related issues were the most common, followed by process problems and some documentation issues. The number of interventions by the author were also comparable across both hackathons, with 18 interventions in the first and 16 in the second. Notably, the first user group required no process-related hints from the observer. In both cases, documentation issues stemmed from missing or insufficiently detailed information. Although

[1] https://github.com/gitpold/master-thesis-mlops/tree/v1.0.0/user-tests

not immediately apparent in this quantitative summary, a review of the full protocols revealed that process and tool-related issues often occurred together within topics. These underlying themes are further explored in Section 7.3.

The results achieved by both user groups during their hackathons differ. The first user group re-implemented the RAG use case described in Section 7.1.1. The participants managed to complete all non-optional instruction steps up to running the fully functional data product locally on a feature branch. However, due to time constraints, they were unable to proceed with the CI/CD component. Specifically, creating the pull request from the feature branch to the `DEV` branch, ensuring the GitHub Actions pipeline runs successfully, and deploying the data product via Terraform was not achieved. The second test user group re-implemented the image classification use case described in Section 7.1.2. This group managed to complete nearly all instruction steps, including optional ones, up to the deployment of the data product on the `DEV` stage. The only aspect not completed within the hackathon's time frame was waiting for the automated deployment of the data product to finish. Although both hackathons had a comparable effective duration of around five hours, the difference in outcomes cannot be directly used to draw conclusions. This is not only because the implemented use cases differ but also because the instructions for the second hackathon are enhanced with more pre-prepared code snippets, based on the time issues observed during the first hackathon.

7.2.3 Responses from feedback rounds

After conducting each hackathon, a concluding feedback session was held. This was conducted as a modified focus group interview with a prepared structure in a digital collaboration medium, as described in Section 4.3.2.

A subsequent review of the recordings revealed that a complete transcription is not necessary. For the first user group, which conducted the hackathon remotely, no additional points or information were found in the recording compared to the notes collected in the collaboration tool. For the second user group, which conducted the hackathon on-site, the recording also included no new points, but it was used to add additional context to the notes in the collaboration tool. Following is, therefore, a summary of the feedback points relating to the MLOps pipeline, structured as they were collected in the collaboration tool.

What did you like?

- Easy to use and template enables fast development
- End-to-end example integrated in one place
- Good visual representation of the process
- Integration of local and remote development on Databricks
- Possibility to optionally adapt steps based on requirements of use case
- Great tracing
- Integration with Unity Catalog

In summary, the positive feedback claims that the MLOps process is well integrated and presented, and that the provision of the template enables easy and rapid development. Additionally, the ability to adapt individual steps in the process according to the use case was viewed positively. The possibility of developing both locally and remotely via Databricks was also highlighted favorably. Furthermore, two functionalities of Databricks were perceived very positively, namely the tracing of LLM calls with MLflow and the integration with the Unity Catalog.

What did you not like?

- Strong dependency on single platform
- Strong focus on notebooks (risk of losing structure)
- Seeing danger of Databricks getting confusing and hard to maintain

Points that participants perceived as negative include the strong dependency on a single platform. The strong focus on notebooks in Databricks was also criticized, as this can make structured development more difficult. Continuing this line of thought, the concern was stated that for larger projects, the overview over the resources in Databricks could become confusing, and the project might become difficult to maintain properly.

How could it be improved?

- Reducing dependency on LangChain library in RAG use case
- Provide more examples
- More code snippets of reoccurring parts for faster development
- More different templates for different types of data products
- Include more documentation on fundamentals
- Include more testing in template
- Add permission concept

As suggestions for general improvement, it was mentioned to include more concrete code examples or snippets in the template, as well as to provide additional templates for different types of data products to enable faster development. Additionally, it was mentioned that more foundational knowledge should be provided overall. Specific improvement points included reducing the dependency on the LangChain library in the RAG template. Also it was requested to more clearly demonstrate in the template, with examples, how testing can be integrated. As a more advanced point, and less related to an individual project but concerning the entire MLOps infrastructure, the need for a permission concept was mentioned.

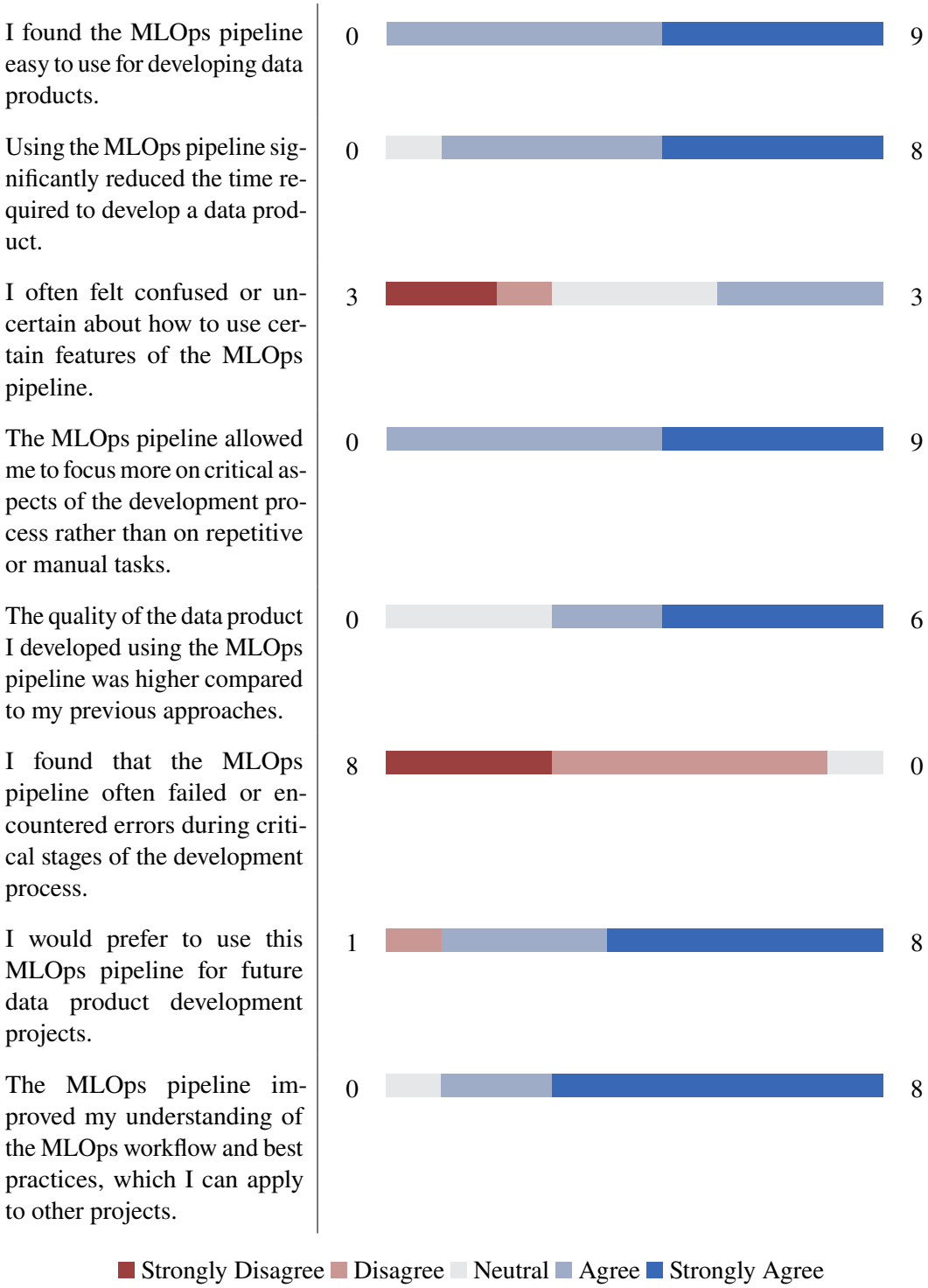


Table 7.5: Results of the anonymous survey.

7.2.4 Result of anonymous survey

In addition to the feedback round, a short anonymous survey was performed after the user tests. This anonymous survey contains eight different questions to be answered on a five point Likert scale as described in Section 4.3.2. The survey was filled out by the in total nine participants combined over both test user groups. The Table 7.5 shows the answer distribution for each of the survey questions. For most of the questions agreement is the positive outcome, however, for question 3 and question 6, disagreement is desired. While in most cases the answers are positive, some answers contain neutral or negative responses. Only the third question stands out, with each three negative, neutral, and positive responses.

The following section Section 4.3.3 describes what can be derived from the responses to this survey as well as the other collected feedback and observations made during the user tests.

7.3 Analysis

The previous section described the execution of the user tests by applying the MLOps pipeline to sample use cases and the collection of the evaluation results shown in Section 7.2. In this section, the results of the hackathons and the individual feedback methods first get analyzed separately, drawing interesting insights from them. For the participant observations, open coding is applied to the list of the encountered problems to identify possible problem areas. Subsequently, the different feedback channels are getting considered in combination to answer research question **RQ3** and assess the effectiveness of the developed MLOps pipeline. Additionally, other interesting insights that can get derived from the collected feedback are presented. Finally, the received improvement suggestions are elaborated.

Comparison of achieved results

A very interesting point arises when comparing the results achieved by both test user groups. Although the two test user groups implemented different sample use cases, some insights can be drawn from comparing the two hackathons. Despite having the same time frame, the first user group was not able to complete the entire MLOps pipeline, while the second user group nearly succeeded. And that even though the MLOps pipeline builds upon the existing DevOps concept from the case study environment, with which the first user group is familiar, but the second user group is not directly acquainted.

One reason for this difference is certainly that, based on the outcome of the first hackathon, the scope and the prepared materials for the second hackathon were adjusted, which naturally reduced complexity and contributed to the second user group successfully going through the entire pipeline. Another reason observed during the hackathons was that the second user group benefited greatly from the diverse experiences of its participants. For each phase of the MLOps pipeline, there was at least one participant who was familiar with the general topic. This difference was particularly evident in the *CI/CD* component. While the first user group faces comparatively high difficulties

with this component due to lack of expertise, the second user group contains two participants with deep expertise in Terraform and GitHub Actions, who were able to guide the group through this step.

This is an example that the application of the MLOps pipeline, or rather how easily it can be applied, depends heavily on the prior knowledge of the participants. While the application seems possible even without deep prior knowledge, the execution of a step is significantly accelerated when at least one person from the test group has prior knowledge and a corresponding level of expertise in this area.

Analysis of test user groups

In addition to the previously described differences between both test groups, other interesting observations were made. For example, it is noteworthy that the first test group, especially when working with Databricks, quickly came up with the idea of looking for and using the corresponding functionality in the Databricks UI when having problems with implementing something via code. This approach was able to speed up the development process, particularly during experimental development. The second test user group, partly due to less experience with Databricks and a personal preference for code-driven development, seemed less inclined to consider achieving results faster through the Databricks UI. One participant even directly asked during the creation of the serving endpoint for testing the model developed on a feature branch about possibilities to create the model serving endpoint via code instead of via the UI. In this context, participants were sometimes not satisfied with how a functionality is implemented and actively looked for alternative solutions, especially when steps had to be executed manually in the Databricks UI.

Because the course of the hackathon to introduce the MLOps pipeline often first showed a simpler, manual way and then via code, this led to participants often doing a later step or an improvement already on their own. An example of this was when creating the model serving endpoint, monitoring was turned on directly, which was planned as a subsequent step in the hackathon.

Another interesting point, even if not directly relevant to this thesis, is that both test user groups actively resorted to use support from AI tools for code generation. Often, when the task is clear, this was done to save time; but sometimes, when a step is unclear, they used AI to potentially obtain a directly working solution. Even if with moderate success, this provided some code snippets. Adjusting these snippets seemed to be preferred over starting completely from scratch. This shows that the MLOps pipeline and the provided templates should potentially be expanded with more code examples or snippets, even if these then have to be adapted depending on the use case, as this relates to the apparently preferred workflow of the participants.

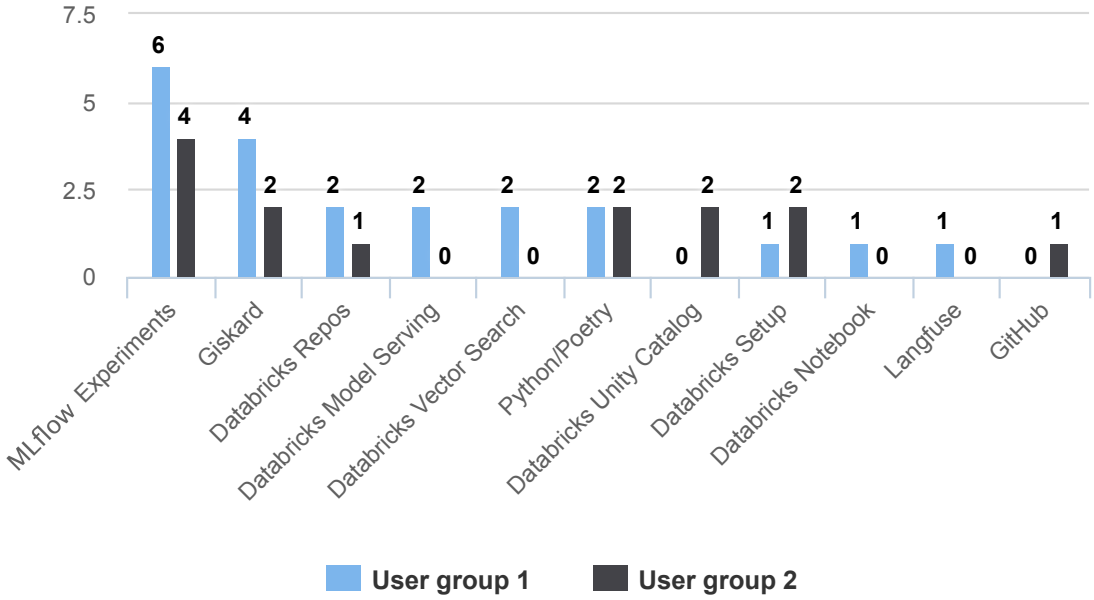


Figure 7.3: Tool-related problems per user group.

Insights from participant observations

In Section 7.2.2, an overview of the number of encountered problems during both hackathons is given. Additionally, the involvement by the observer is described and categorized by different types of problems. For further evaluation of these observations, open coding was applied to the problem categories *tool* and *process*, as described in Section 4.3.3. Although this coding was carried out retrospectively, it was to a certain extent already included due to the context and the detailed descriptions of observations or the given answers in the case of an intervention by the observer, and can be easily performed based on these information.

The problem clusters identified for tool-related problems, are shown in Figure 7.3. For the process-related problems Figure 7.4 provides an overview of the identified clusters.

It must be noted that the frequency of how often a problem area was encountered does not necessarily indicate that this area led to more problems than other tools or MLOps components, as these may have been just less intensively addressed in the user tests. Since experiment tracking with MLflow represented a central component in both use cases, it was expected to a certain extent that more problems occur with it.

Still, from this identification of problem areas, it becomes apparent for which tools or parts of the MLOps pipeline further information and assistance should be provided. This concerns, for example, MLflow, particularly the experiment tracking component, and Giskard, as well as some other functions within Databricks. While this can be explained by the fact that the tools are unfamiliar for many participants, it must be assumed that future users may also be unfamiliar with them. Adjustments by providing additional documentation should, thus, be made accordingly.

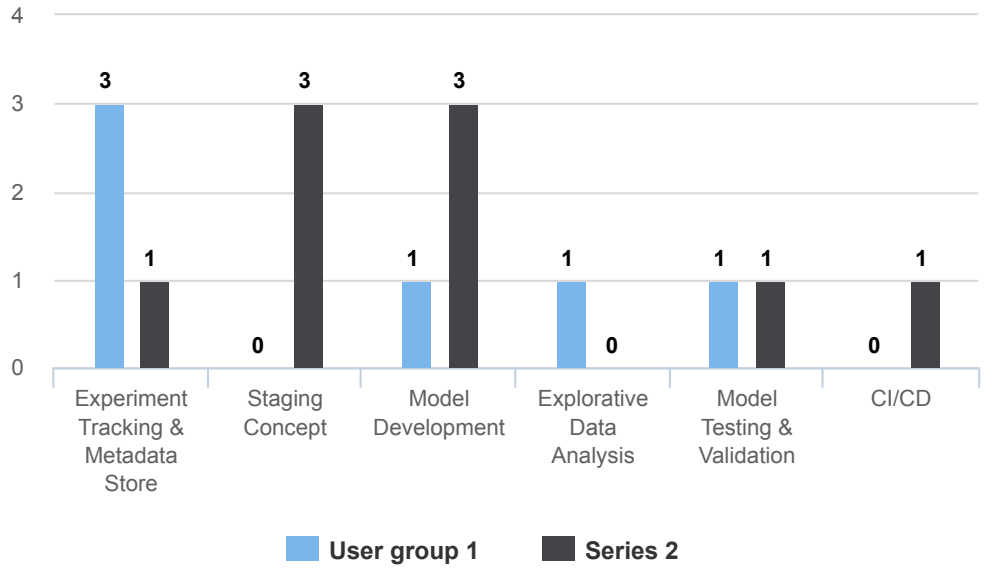


Figure 7.4: Process-related problems per user group.

Furthermore, it is noticeable that both user groups experienced problems with the basic setup and configuration of Python and Poetry. This is evidently due to the author's assumption that familiarity with and use of these technologies can be presupposed, which turned out to be incorrect in the user tests and is, therefore, likely to be the case with future users as well. Accordingly, the initial setup of the template and its dependencies should be better described.

From a process perspective, the problems with model development are noteworthy. This is another point that was implicitly assumed as a prerequisite in the design of the MLOps pipeline. If this MLOps pipeline is to be used by users without prior experience in model development, appropriate extensions of the documentation are definitely necessary.

It is also interesting that the second test user group had some problems with the staging concept and overall structure of the MLOps pipeline, whereas this does not occur with the first user group. This could be due to the fact that the process originates from the existing DevOps process with which the first user group is already familiar, or it could be because the first group spent less time on this topic due to time constraints.

Insights from anonymous survey

Since the provided survey was completed by all participants of the user tests, but thus only by nine people, the significance and representativeness of the survey is limited. In addition to the small number of participants, all participants personally know the author of this work and may have potentially not provided a completely objective opinion. An evaluation of the results based on absolute answers is therefore not very meaningful. However, the differing levels of agreement between the individual questions can be assessed.

Statements 1, 2, 4, 6, 7, and 8 each received positive agreement from at least eight of the nine survey participants, as seen in Table 7.5. From this, it can be concluded that the aspects of simplicity and acceleration of data product development with this MLOps pipeline are fulfilled to a certain extent. Additionally, it seems to be the case that the pipeline increased the understanding of the MLOps workflow and concept itself.

More significant and interesting are the questions that received comparatively less agreement. These highlight the points that were least well implemented and have the most potential for improvement. This includes, as described in Section 7.2.4, question 3, which has equal levels of agreement and disagreement, suggesting that the process and features of the MLOps pipeline are not sufficiently described and explained. Also, question 5, which did not receive disagreement but comparatively less agreement, suggests that this aspect is improvable. Although the goal of this work was not directly defined as to to increase the quality in the development of data products, it is certainly relevant for the application of this MLOps pipeline that the achieved results are at least as good as with comparable approaches.

Research Question

The various feedback methodologies highlight some issues with the current version of the MLOps pipeline, such as the recurring suggestion that more documentation and code examples should be included.

Considering the predominantly positive survey results, the numerous affirmative responses from the feedback session, and the near replication of the hackathon's target outcome within approximately six hours, the designed MLOps pipeline, along with the template, undeniably simplifies the development of data products. Implementing a complete use case—even if only an example—within this timeframe signifies a substantial improvement over the expected time investment required for a manual implementation.

However, the question posed in research question **RQ3** regarding the effectiveness of the MLOps pipeline is difficult to answer based on the evaluation results, as there is no appropriate comparative benchmark. While some participants already had experience with the development of data products, others did not, which makes the already subjective assessments even more challenging to interpret. Similarly, demonstrating a qualitative improvement in the development process is difficult for the same reasons, even though two-thirds agreed with the corresponding statement in the anonymous survey.

Improvement Potential

During the execution of the hackathons and from the collected feedback, several areas for improvement emerged. The feedback responses contain concrete, actionable points, such as reducing the dependency on a particular library, but also general points, especially regarding documentation and the provided templates.

This improvement potential regarding documentation is evidenced by the process-related problems and discussions about how to continue next during the hackathon, as well as by the most negatively rated statement in the anonymous survey, which suggests that some participants were often uncertain or confused about features of the MLOps pipeline.

A major point is also the provisioning of more documentation on fundamental topics. This can also be concluded from the participant observations, as participants with little or no prior experience in the development of ML models were unfamiliar even with basic terms and concepts related to ML model development, which had, however, been assumed for using this MLOps pipeline. While not all terms and concepts can be explained within the template for this MLOps pipeline, it was suggested to supplement the template with a glossary and to provide links to further explanations of concepts.

Perhaps the most significant point, which comes up in various ways during the feedback round, is that while the provided templates are very good, they should contain more examples, especially concrete code examples. These could then be applied directly, for example, for recurring functions, leading to even faster development. This was also reflected in the behavior of both user groups, who often consulted a LLM for a solution when no starting point was available and then iterated and adapted on top of this response. This feedback strongly aligns with the results from the observation of the hackathons.

Ultimately, many of the problems that arose stem from participants not being familiar with a tool and the documentation not being sufficient. Although starting points were prepared for the hackathon and documentation for the respective tools was provided, extracting the relevant code parts required a deeper understanding, leading to more effort and taking correspondingly longer. By providing ready-made snippets or concrete examples that only need minimal adjustment, time could be saved.

However, it must always be weighed what makes sense as a general part in every project for the development of data products and what is too specific. After all, while the MLOps pipeline is intended to accelerate development, providing a large number of ready-made templates for many use cases increases maintenance effort and goes beyond the intended scope. Nevertheless, this might be a valid approach, especially for users with limited ML experience, in order to enable them also to effectively develop data products.

8 Discussion

This chapter discusses the findings of the thesis and contributes valuable insights regarding the overarching research question:

How can an MLOps pipeline be leveraged to improve the development and operational processes of data products?

Due to the division of this thesis into three research goals, the achievement and insights of each goal are assessed and discussed separately.

- Goal 1: Evaluation of MLOps approaches (Section 8.1)
- Goal 2: Design and implementation of a data product development pipeline (Section 8.2)
- Goal 3: Evaluation of the data product development pipeline (Section 8.3)

Furthermore, Section 8.4 discusses the limitations of the solution and the threats to its validity. Since this thesis takes place in a case study environment, it is particularly relevant to discuss to what extent the solution is generalizable and how reliably insights and results can be transferred to other cases.

8.1 Evaluation of Machine Learning Operations approaches

For the phase related to the first research goal (Goal 1), both research questions, **RQ1** and **RQ2**, have been addressed. Additionally, interesting findings regarding the comparison of the defined MLOps component model and challenges during the identification of platforms and tools are discussed.

Regarding **RQ1**, architectural options were explored, including whether to use MLOps platforms and different model propagation concepts. This phase also identified and prioritized a comprehensive list of MLOps components based on the Rapid Review. The prioritization indicates which components are most relevant for the case study environment but also offers general insights into which components should be addressed first when adopting an MLOps concept. The second research question, **RQ2**, was answered by comparing various platforms and tools for the identified MLOps components. By evaluating these options based on the specific requirements of the case study, it was possible to determine which tools were suitable or unsuitable for the pipeline. These comparisons also provide valuable insights that can be applied to other case study environments with different requirements.

Comparing the defined MLOps components in this thesis to the component models by Kreuzberger et al. and Valohai revealed similarities and differences. All components mentioned by Kreuzberger et al. are included, though some have slightly different names or are consolidated, such as the combination of *Model Training Pipeline & Workflow Orchestration*. Additionally, components

not explicitly covered by Kreuzberger et al., like *Data Labeling*, *Data Versioning & Management*, *Exploratory Data Analysis*, *Hyperparameter Tuning*, and *Model Testing & Validation*, are included in this definition. A similar trend emerges when compared to the MLOps stack defined by Valohai. While all core components are included, the two separately listed components, *Data Analysis* and *Experimentation*, are consolidated into *Exploratory Data Analysis*. Components from this definition that do not appear in Valohai's stack include *Data Labeling*, *Data Versioning & Management*, *CI/CD*, *Hyperparameter Tuning*, and *Model Testing & Validation*.

The differences between the models can be explained by several factors. First, defining the boundaries between components is often difficult, as components can either be listed separately or seen as part of another component. For example, the *Hyperparameter Tuning* component could be viewed as part of the model development process and, therefore, might not appear as a separate component. Additionally, this definition includes aspects of LLMOps, leading to the inclusion of *Model Testing & Validation* as a standalone component, which is particularly relevant for LLMs. Moreover, the two presented component models do not include a component for data preprocessing, assuming a feature store with processed data is already available. In the case of Kreuzberger et al., however, the data preprocessing is still covered by a data engineering zone as part of an end-to-end MLOps architecture combining the components with an overall MLOps workflow [KKH23]. In this case, while there is also no separate component for data preparation, there are distinct components for data versioning and labeling. The absence of a data preparation component in this definition, as seen in some MLOps phase-based views, is because this phase often overlaps with the code for model development and employs no standalone component.

The process of identifying platforms and tools during the Rapid Review revealed challenges, particularly due to inconsistencies in how different sources classify technologies in the MLOps field. These inconsistencies often stem from ambiguities in the literature, particularly in grey literature. Some sources describe certain technologies as comprehensive MLOps platforms, while others classify them as tools that only address specific components of the pipeline. This lack of clarity is problematic for users trying to determine which tools meet their needs. Several factors contribute to this:

- Component-wise comparison: Some sources compare individual components of the MLOps workflow but include both specialized tools and the respective parts of platforms. This leads to platforms being named in the context of a single component. In this case, however, the platform should then be named for each component it covers, providing some insights on which components a platform does or, more importantly, does not cover.
- Terminology variations: The term *platform* is used inconsistently across the literature. What one source defines as a complete MLOps platform may be seen by another as a tool that only covers a portion of the overall pipeline. Some sources distinguish here between simple MLOps platforms covering multiple components and full end-to-end platforms spanning the whole MLOps cycle.
- Marketing influences: Many technologies market themselves as comprehensive platforms, although they may only partially cover the MLOps lifecycle. Some may simply integrate multiple tools, offering a broader service, but not necessarily a complete end-to-end solution.

Furthermore, some tools cover multiple components, which can blur the lines between their primary functionality and additional features. For example, tools like Arize Phoenix support both *Model Testing & Validation* and *Monitoring & Observability*, offering synergies when using them but making it more complicated to categorize and compare them within a rigid component framework.

Goal 1: Evaluation of MLOps approaches

RQ1: Through the Rapid Review, 13 components were synthesized to define a component model that is sufficient for most MLOps projects. Exempting components already known through the DevOps approach, *Data Versioning & Management*, *Experiment Tracking & Metadata Store*, *Model Testing & Validation*, *Model Registry*, and *Model Deployment & Serving* were identified as most relevant. Additionally, different model deployment concepts have been compared, and using an MLOps platform extended by single tools has emerged as a promising approach.
RQ2: In total, 6 MLOps platforms and 59 tools (some of them part of the platforms) were evaluated for the different MLOps components, identifying Azure ML and Databricks as promising platforms and many additional tools suitable for this case study environment.

8.2 Design and Implementation of a Data Product Development Pipeline

Regarding the second research goal, the objective of designing an MLOps pipeline (Goal 2) has been successfully achieved. While some individual components are not fully defined by predetermined tool choices, all relevant components of an MLOps pipeline have been defined. The necessary infrastructure components have been implemented, ensuring the direct application of the MLOps pipeline, for instance, through the provided templates.

Moreover, the goal of tailoring the MLOps pipeline to the context of the case study environment has been met. The defined requirements were applied not only in the selection of tools but also in the overall design of the MLOps pipeline. The developed solution meets all defined constraints and all directly measurable requirements, except for the support of on-premise use cases. As this requirement had a low priority and complicated the fulfillment of other higher-priority requirements, it was deprioritized in alignment with the responsible stakeholders in the case study department. However, from a technical perspective, this requirement could still be fulfilled later by implementing a direct connection from Azure to the company's on-premise network, albeit with considerable effort. Additionally, the solution provides a strong foundation for meeting the non-directly measurable criteria, particularly those with the highest priority, such as maintainability and user experience.

The implementation of the MLOps pipeline, along with setting up the infrastructure and the Databricks workspace using Terraform, was successful, although the effort required, especially regarding network architecture and security, was greater than initially anticipated. These findings further highlight the relevance of IaC setups, as they simplify maintenance and facilitate reuse of the setup in other areas.

Goal 2: Design and implementation of a data product development pipeline

DP1: The complete infrastructure required for the MLOps pipeline has been implemented via Terraform. The developed solution integrates with existing infrastructure on Azure and DevOps processes using GitHub and Terraform. The pipeline is built on the Databricks platform, leveraging integrated components such as Unity Catalog and MLflow. It is extended with additional tools like Langfuse and libraries like Giskard and Ragas. Template repositories and a usage guide have been provided to facilitate easy adoption of the MLOps pipeline.

8.3 Evaluation of the Data Product Development Pipeline

For the phase related to the third research goal (Goal 3), the achievement of its research question **RQ3** is considered. Additionally, challenges regarding enabling less technically experienced users to use the MLOps pipeline are discussed. Finally, an assessment of the overall evaluation outcome is provided.

The goal of this phase, to improve the development process of data products, was achieved to some extent. However, the exact measurement of this improvement was challenging due to the nature of the evaluation conducted in this thesis. In particular, a quantitative assessment was not feasible due to the lack of direct comparisons. Nevertheless, based on the observations, it can be reasonably assumed that using the MLOps pipeline, alongside the provided templates, significantly accelerates data product development. Both test user groups were able to develop and nearly deploy a data product within a single day. Although the process involved a prepared use case and support from the observer, it is likely that, in an isolated setup, a similar result could be achieved within a few days. This still represents a short development time within the case study context. A more detailed user test with a specific focus on evaluating individual components could have provided more insights but was not feasible within the time constraints of this thesis.

Regarding the requirement that users with less technical experience should also be able to develop a data product using this MLOps pipeline, the following observations were made: In its current form, the entry barrier for users without prior experience was too high, and the supplementary and introductory documentation provided was insufficient to bridge this gap fully. However, for users with prior technical experience, using the MLOps pipeline was feasible.
The hackathons demonstrated that participants with deep expertise in specific areas of the pipeline could apply their knowledge efficiently, completing tasks in significantly less time. While this was expected, it indicates potential for improvement, particularly in making the pipeline more accessible to users with intermediate or beginner-level experience.
This was further reflected in the feedback round, where many improvement suggestions focused on providing more comprehensive documentation and, especially, directly usable code examples. For broader application of the MLOps pipeline, especially if used as a blueprint in other areas, a structured concept for introducing the pipeline should be developed. This could involve simplifying specific components, expanding the documentation, providing more prepared templates, or offering additional training and support to guide users through the pipeline.

Furthermore, while this thesis focused on the application of the pipeline, users did not test the setup and operation of the required infrastructure. Although the infrastructure was implemented using an IaC approach with Terraform, it remains unclear how easily users with little experience in cloud infrastructure could deploy and maintain it. While it is expected that users with experience in Terraform and Azure should manage the setup efficiently, this assumption would need to be validated with more extensive testing.

Goal 3: Evaluation of the data product development pipeline

RQ3: The MLOps pipeline provides a solid foundation for accelerating data product development, particularly for users with some previous technical experience. The test users were able to develop and deploy data products within a short time frame of one day, proving the pipeline's effectiveness in practical scenarios. However, for less experienced users, it is challenging due to insufficient documentation. Expanding the documentation and including more detailed examples and templates would enhance usability. Finally, more extensive testing with a broader user base and in larger use cases is necessary to fully evaluate the performance of the MLOps pipeline.

8.4 Threats to Validity

The development of the data product development pipeline in this thesis involved several stages: gathering requirements, conducting a theoretical evaluation of MLOps approaches and tools, designing and implementing an MLOps pipeline, and applying and evaluating it in user tests with sample use cases. This comprehensive scope, combined with time constraints, introduces several limitations and potential threats to validity that were either mitigated or accepted in the pursuit of the overall goal.

For the evaluation of MLOps approaches, a Rapid Review was employed. While this method follows standard criteria for literature selection, its inherent time-boxed nature may have excluded sources that could have been captured in an SLR. Additionally, due to the rapid development of the MLOps field, grey literature was included, posing the risk that some of the included sources provided outdated, imprecise, or wrong information. This was mitigated by transparently documenting all architectural decisions and tool selections through ADRs.

Furthermore, while the platform and tool identification were based on the Rapid Review results, there were too many for a detailed evaluation of all of them. Although the selection for the detailed evaluation is thoroughly justified, the abstinence of a formal selection methodology, such as a weighted decision process, represents a limitation. Some potentially suitable tools and platforms may have been omitted from further evaluation.

During the requirement analysis, it was recognized that the non-functional requirements, as defined in Section 4.1.1, could not be directly measured. Ideally, measurable criteria for these requirements would have been determined from the start to facilitate a more straightforward evaluation. However, due to the complexity of the overall system, it was not feasible to establish precise measurable criteria at that time. Instead, the non-functional requirements were treated as general qualities to be prioritized during the development of the MLOps pipeline. This introduces some ambiguity in evaluating how well these requirements were ultimately met.

Another issue relates to the ongoing development of tools in the MLOps space. Many tools evolve rapidly, with new versions being released frequently. Consequently, the evaluated tools may have changed, and their suitability for the defined components might shift with future updates. Additionally, there can be free and paid versions of a tool with different feature sets, which introduces uncertainty as to which version a literature source refers to. Another key challenge in tool and platform identification was the inconsistency across sources in classifying MLOps technologies. Different sources referred to technologies as either platforms or tools, depending on how broadly they covered the MLOps lifecycle. This ambiguity, coupled with marketing influences that tend to overstate the capabilities of certain tools, added complexity to the comparison process. Some tools were difficult to categorize, as they spanned multiple components. For example, tools like Arize Phoenix cover both *Model Testing & Validation* and *Monitoring & Observability*, making it harder to assign them to a single component.

The evaluation of the developed MLOps pipeline faced certain limitations due to the scope of the user tests. Only two test groups, with a total of nine participants, were involved, limiting the breadth of feedback and the statistical significance of the findings. Furthermore, observer involvement was allowed during the hackathons, which could have influenced the results. Although these interventions were documented and mainly intended to save time to complete the tests within the limited timeframe, it is possible that user performance and feedback would have been different in a fully independent setup. This potential bias is important, as a test group without access to the observer may have encountered more difficulties and come to less favorable outcomes.

Moreover, this thesis faces a significant threat to external validity. The data product development pipeline was designed and implemented with a strong influence from the case study environment within a corporate department with specific constraints. These constraints included the need to support existing systems and processes, leading to a bias in technology selection.

This limitation could not be entirely circumvented within the framework of this thesis. The optimal MLOps pipeline for a different environment with other constraints and requirements can and will differ from the one developed here. However, the approach presented in this thesis goes beyond merely describing the final MLOps pipeline. It covers the entire process—from defining MLOps components and comparing tools to designing, developing, and evaluating the pipeline's effectiveness. This approach itself is generalizable and can be applied to other case study environments. While the scope of tools in the comparison would need to be expanded based on initial requirements, the MLOps components and general insights into architecture, design, and implementation can be adopted. Building upon this, another data product development pipeline can be developed using this work as a reference in a different case study environment.

Furthermore, it should be noted that while the developed MLOps pipeline will not be the optimal solution in every case, it is still applicable in many scenarios—particularly if there are no significant objections to using Databricks as the MLOps platform. Databricks and Langfuse are not only compatible with Azure but can also be used seamlessly on Amazon Web Services and Google Cloud Platform. If GitHub is also in use, all infrastructural requirements for operating the developed MLOps pipeline are already met, although adjustments might be needed depending on the cloud provider.

9 Conclusion and Outlook

This thesis outlines the complete process of developing an MLOps pipeline aimed at enhancing the development and operation of data products. Just as this work is divided into three distinct phases, the achieved results are structured accordingly.

In the first phase, a comprehensive evaluation of various MLOps approaches was undertaken. This included an extensive comparison of different MLOps platforms and tools based on a Rapid Review, alongside the initial requirements analysis. This phase not only laid the foundation for the following phase but also serves as a valuable informational basis for future research in this field.

The second phase focused on the design and implementation of an MLOps pipeline. A functional pipeline was developed, tailored to the specific requirements and constraints of the case study environment while incorporating practical insights gained during the implementation process. The complete MLOps infrastructure was deployed using IaC, and different templates were provided for the development of data products, pre-configured with the necessary setup and documentation to apply the MLOps pipeline. As a result, a fully customized and functional MLOps pipeline was not only designed but also implemented and prepared for practical use. This pipeline seamlessly integrates into the existing infrastructure and DevOps processes defined by Azure and GitHub and is based on the Databricks platform. Ultimately, the pipeline leverages tools such as Unity Catalog with the Delta Lake data format, MLflow for tracking and model registry, Ragas and Giskard for model and LLM response validation, Langfuse for LLM tracing, and other internal Databricks components.

The third phase involved evaluating the developed MLOps pipeline. It was tested by users during two practical hackathons, where its effectiveness in developing data products was assessed. This evaluation provided not only insights into the pipeline's performance and successful application but also highlighted areas for improvement, such as the need for enhanced documentation.

In conclusion, this thesis goes beyond just designing an MLOps pipeline; it encompasses the complete development process and its evaluation in a practical setting. It addresses a gap in the literature where many studies focus on tool comparisons or pipeline design but seldom evaluate these solutions in real-world environments. This work offers valuable insights for companies and teams, presenting a concrete MLOps pipeline set in a typical corporate setup (Azure and GitHub). It provides a practical approach to adopting an MLOps concept, offering not only a theoretical framework but also direct input on what works well in practice and which areas require further attention and refinement for successful real-world application.

9.1 Future Work

Although this thesis has developed a complete MLOps pipeline that can be used as-is for the development of data products, additional topics can be addressed in future work.

One natural option is the further development of the MLOps pipeline. This includes, for example, the definition of the remaining components that were not fully defined due to their lower priority, as well as the expansion of existing components or the entire workflow. While Giskard and Ragas are used to evaluate security metrics for LLMs, the MLOps pipeline could be extended to focus more on previously less emphasized aspects such as bias and fairness metrics or security and guardrails for LLMs.

Furthermore, the user feedback can be incorporated into the MLOps pipeline. In particular, the topic of more extensive documentation can be addressed by designing training sessions or introductions to this pipeline and the general application of MLOps. An interesting approach could be to provide a chatbot, as demonstrated in this thesis, which offers information about the MLOps pipeline itself. Alternatively, the approach of providing templates, touched upon in this work, could be further pursued. In future work, various templates could be designed that are easily usable even for users with little to no experience, thereby including not only the pipeline but also pre-built components needed for the majority of data products. Taking this idea further, a wizard for a configurable template could be provided. With these steps completed, everything would then be in place for a potential rollout of the solution to other areas of the company.

For the two aforementioned proposals, as well as for the MLOps pipeline developed in this work, more extensive user tests could be conducted—for example, with more test users, and under even more practical conditions, with more time and separate control groups. Additionally, the validation of the pipeline could be investigated using a real, larger use case. Furthermore, not only the application of the MLOps pipeline but also the setup and operation of the required infrastructure could be tested.

As already described, the data product development pipeline developed within the scope of this thesis heavily depends on the requirements and constraints of the case study environment. Therefore, it could be examined how the MLOps pipeline can be applied in other departments under potentially different conditions. Extending this idea further, the approach described in this work for developing the MLOps pipeline could be applied to completely different case study environments in other companies. This would allow for verification of this approach and enable the MLOps concept and advantages of an MLOps pipeline for the development of data products in additional areas.

Bibliography

[AA21] S. Alla, S. K. Adari. “What Is MLOps?” en. In: *Beginning MLOps with MLFlow*. Berkeley, CA: Apress, 2021, pp. 79–124. ISBN: 978-1-4842-6548-2 978-1-4842-6549-9. DOI: 10.1007/978-1-4842-6549-9_3. URL: http://link.springer.com/10.1007/978-1-4842-6549-9_3 (visited on 05/20/2024) (cit. on pp. 23, 24, 29).

[ABB+19] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, T. Zimmermann. “Software Engineering for Machine Learning: A Case Study”. In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 2019, pp. 291–300. DOI: 10.1109/ICSE-SEIP.2019.00042 (cit. on p. 20).

[ASS+24] M. Ashley, N. Solomon, S. Sawyerr, K. Junčytė, B. Washington. *DevOps - the web’s largest collection of DevOps content*. Feb. 2024. URL: https://devops.com/ (cit. on p. 22).

[BJS19] L. Baier, F. Jöhren, S. Seebacher. “Challenges in the Deployment and Operation of Machine Learning in Practice”. en. In: (2019). Medium: PDF Publisher: [object Object]. DOI: 10.5445/IR/1000095028. URL: https://publikationen.bibliothek.kit.edu/1000095028 (visited on 05/18/2024) (cit. on pp. 15, 29).

[BKTT22] J. Bradley, R. Kurlansik, M. Thomson, N. Turbitt. *The Big Book of MLOps*. Databricks, 2022 (cit. on pp. 50, 53, 54, 80).

[Bos14] J. Bosch. “Continuous Software Engineering: An Introduction”. en. In: *Continuous Software Engineering*. Ed. by J. Bosch. Cham: Springer International Publishing, 2014, pp. 3–13. ISBN: 978-3-319-11282-4 978-3-319-11283-1. DOI: 10.1007/978-3-319-11283-1_1. URL: https://link.springer.com/10.1007/978-3-319-11283-1_1 (visited on 07/21/2024) (cit. on p. 22).

[BWZ15] L. Bass, I. Weber, L. Zhu. *DevOps: a software architect’s perspective*. eng. The SEI series in software engineering. New York Boston Indianapolis San Francisco Toronto Montreal London Munich Paris Madrid Capetown Sydney Tokyo Singapore Mexico City: Addison-Wesley, 2015. ISBN: 978-0-13-404988-5 978-0-13-404984-7 (cit. on p. 23).

[CPS20] B. Cartaxo, G. Pinto, S. Soares. “Rapid Reviews in Software Engineering”. In: *Contemporary Empirical Methods in Software Engineering*. Ed. by M. Felderer, G. H. Travassos. Cham: Springer International Publishing, 2020, pp. 357–384. ISBN: 978-3-030-32489-6. DOI: 10.1007/978-3-030-32489-6_13. URL: https://doi.org/10.1007/978-3-030-32489-6_13 (cit. on pp. 31, 32).

[EAN+17] F. Elberzhager, T. Arif, M. Naab, I. Süß, S. Koban. "From Agile Development to DevOps: Going Towards Faster Releases at High Quality – Experiences from an Industrial Context". en. In: *Software Quality. Complexity and Challenges of Software Engineering in Emerging Technologies*. Ed. by D. Winkler, S. Biffl, J. Bergsmann. Vol. 269. Series Title: Lecture Notes in Business Information Processing. Cham: Springer International Publishing, 2017, pp. 33–44. ISBN: 978-3-319-49420-3 978-3-319-49421-0. DOI: 10.1007/978-3-319-49421-0_3. URL: https://link.springer.com/10.1007/978-3-319-49421-0_3 (visited on 07/21/2024) (cit. on pp. 22, 23).

[FS17] B. Fitzgerald, K.-J. Stol. "Continuous software engineering: A roadmap and agenda". In: *Journal of Systems and Software* 123 (2017), pp. 176–189. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2015.06.063. URL: https://www.sciencedirect.com/science/article/pii/S0164121215001430 (cit. on pp. 22, 23).

[Gar20] Gartner. *Gartner Identifies the Top Strategic Technology Trends for 2021*. 2020. URL: https://www.gartner.com/en/newsroom/press-releases/2020-10-19-gartner-identifies-the-top-strategic-technology-trends-for-2021 (cit. on p. 15).

[GFM19] V. Garousi, M. Felderer, M. V. Mäntylä. "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering". In: *Information and Software Technology* 106 (2019), pp. 101–121. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2018.09.006. URL: https://www.sciencedirect.com/science/article/pii/S0950584918301939 (cit. on p. 29).

[GFMR20] V. Garousi, M. Felderer, M. V. Mäntylä, A. Rainer. "Benefitting from the Grey Literature in Software Engineering Research". In: *Contemporary Empirical Methods in Software Engineering*. Cham: Springer International Publishing, 2020, pp. 385–413. ISBN: 9783030324896. DOI: 10.1007/978-3-030-32489-6_14 (cit. on p. 32).

[Goo] Google. *Artificial intelligence (AI) vs. machine learning (ML)*. URL: https://cloud.google.com/learn/artificial-intelligence-vs-machine-learning (cit. on p. 20).

[GXG+24] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, H. Wang. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2024. arXiv: 2312.10997 [cs.CL]. URL: https://arxiv.org/abs/2312.10997 (cit. on p. 21).

[JOB21] M. M. John, H. H. Olsson, J. Bosch. "Towards MLOps: A Framework and Maturity Model". In: *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. Palermo, Italy: IEEE, Sept. 2021, pp. 1–8. ISBN: 978-1-66542-705-0. DOI: 10.1109/SEAA53835.2021.00050. URL: https://ieeexplore.ieee.org/document/9582569/ (visited on 05/20/2024) (cit. on p. 29).

[KBL08] J. Kontio, J. Bragge, L. Lehtola. "The Focus Group Method as an Empirical Tool in Software Engineering". en. In: *Guide to Advanced Empirical Software Engineering*. Ed. by F. Shull, J. Singer, D. I. K. Sjøberg. London: Springer London, 2008, pp. 93–116. ISBN: 978-1-84800-043-8 978-1-84800-044-5. DOI: 10.1007/978-1-84800-044-5_4. URL: http://link.springer.com/10.1007/978-1-84800-044-5_4 (visited on 08/03/2024) (cit. on pp. 41, 42).

[Kit04] B. Kitchenham. "Procedures for performing systematic reviews". In: *Keele, UK, Keele University* 33.2004 (2004), pp. 1–26 (cit. on p. 34).

[KKH23] D. Kreuzberger, N. Kühl, S. Hirschl. "Machine Learning Operations (MLOps): Overview, Definition, and Architecture". In: *IEEE Access* 11 (2023), pp. 31866–31879. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2023.3262138. URL: https://ieeexplore.ieee.org/document/10081336/ (visited on 05/17/2024) (cit. on pp. 22, 24, 26, 29, 49, 103, 104).

[KSKG23] A. Kulkarni, A. Shivananda, A. Kulkarni, D. Gudivada. "LLMs for Enterprise and LLMOps". en. In: *Applied Generative AI for Beginners*. Berkeley, CA: Apress, 2023, pp. 117–154. ISBN: 978-1-4842-9993-7 978-1-4842-9994-4. DOI: 10.1007/978-1-4842-9994-4_7. URL: https://link.springer.com/10.1007/978-1-4842-9994-4_7 (visited on 05/20/2024) (cit. on pp. 27, 30).

[LCB20] L. E. Lwakatare, I. Crnkovic, J. Bosch. "DevOps for AI – Challenges in Development of AI-enabled Applications". In: *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. Split, Croatia: IEEE, Sept. 2020, pp. 1–6. ISBN: 978-953-290-099-6. DOI: 10.23919/SoftCOM50211.2020.9238323. URL: https://ieeexplore.ieee.org/document/9238323/ (visited on 05/18/2024) (cit. on pp. 15, 29).

[Mat17] J. T. B. Mattivi B. "Rapid Reviews: Sisyphos' Erlösung? – Eine Bestandsaufnahme". DE. In: *Das Gesundheitswesen* 79.05 (June 2017). Edition: 2015/05/18 Publisher: © Georg Thieme Verlag KG, pp. 438–442. ISSN: 0941-3790. DOI: 10.1055/s-0035-1548857. URL: http://www.thieme-connect.com/products/ejournals/abstract/10.1055/s-0035-1548857 (cit. on p. 32).

[Mez] S. Mezak. *The Origins of DevOps: What's in a Name?* URL: https://devops.com/the-origins-of-devops-whats-in-a-name/ (cit. on p. 22).

[MG22] B. M. A. Matsui, D. H. Goya. "MLOps: five steps to guide its effective implementation". en. In: *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*. Pittsburgh Pennsylvania: ACM, May 2022, pp. 33–34. ISBN: 978-1-4503-9275-4. DOI: 10.1145/3522664.3528611. URL: https://dl.acm.org/doi/10.1145/3522664.3528611 (visited on 05/20/2024) (cit. on p. 29).

[MRL+23] S. Moreschi, G. Recupito, V. Lenarduzzi, F. Palomba, D. Hastbacka, D. Taibi. *Toward End-to-End MLOps Tools Map: A Preliminary Study based on a Multivocal Literature Review*. arXiv:2304.03254 [cs]. Apr. 2023. URL: http://arxiv.org/abs/2304.03254 (visited on 05/18/2024) (cit. on pp. 22, 29, 30).

[Mur22] K. P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. URL: probml.ai (cit. on p. 20).

[Pat12] D. J. Patil. *Data jujitsu: the art of turning data into product*. eng. OCLC: 803460018. Sebastopol, CA: O'Reilly, 2012. ISBN: 978-1-4493-4114-5 (cit. on p. 19).

[PSSW21] R. Panikkar, T. Saleh, M. Szybowski, R. Whiteman. *Operationalizing machine learning in processes*. 2021. URL: https://www.mckinsey.com/capabilities/operations/our-insights/operationalizing-machine-learning-in-processes (cit. on p. 15).

[Rig21] X. G. Rigol. *Data as a product vs data products. What are the differences?* 2021. URL: https://towardsdatascience.com/data-as-a-product-vs-data-products-what-are-the-differences-b43ddbb0f123 (cit. on p. 19).

[RMRO21] P. Ruf, M. Madan, C. Reich, D. Ould-Abdeslam. “Demystifying MLOps and Presenting a Recipe for the Selection of Open-Source Tools”. en. In: *Applied Sciences* 11.19 (Sept. 2021), p. 8861. ISSN: 2076-3417. DOI: 10.3390/app11198861. URL: https://www.mdpi.com/2076-3417/11/19/8861 (visited on 05/18/2024) (cit. on pp. 24, 25, 29).

[RPC+22] G. Recupito, F. Pecorelli, G. Catolino, S. Moreschini, D. D. Nucci, F. Palomba, D. A. Tamburri. “A Multivocal Literature Review of MLOps Tools and Features”. In: *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. Gran Canaria, Spain: IEEE, Aug. 2022, pp. 84–91. ISBN: 978-1-66546-152-8. DOI: 10.1109/SEAA56994.2022.00021. URL: https://ieeexplore.ieee.org/document/10011505/ (visited on 05/20/2024) (cit. on pp. 29, 30, 32).

[Sea08] C. B. Seaman. “Qualitative Methods”. en. In: *Guide to Advanced Empirical Software Engineering*. Ed. by F. Shull, J. Singer, D. I. K. Sjøberg. London: Springer London, 2008, pp. 35–62. ISBN: 978-1-84800-043-8 978-1-84800-044-5. DOI: 10.1007/978-1-84800-044-5_2. URL: http://link.springer.com/10.1007/978-1-84800-044-5_2 (visited on 08/03/2024) (cit. on p. 41).

[Sko20] H. Skogström. *The MLOps Stack*. 2020. URL: https://valohai.com/blog/the-mlops-stack/ (cit. on pp. 26, 27).

[SNKP22] G. Symeonidis, E. Nerantzis, A. Kazakis, G. A. Papakostas. “MLOps - Definitions, Tools and Challenges”. In: *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*. Las Vegas, NV, USA: IEEE, Jan. 2022, pp. 0453–0460. ISBN: 978-1-66548-303-2. DOI: 10.1109/CCWC54503.2022.9720902. URL: https://ieeexplore.ieee.org/document/9720902/ (visited on 05/20/2024) (cit. on pp. 24, 25, 29, 52).

[SSV22] R. Subramanya, S. Sierla, V. Vyatkin. “From DevOps to MLOps: Overview and Application to Electricity Market Forecasting”. In: *Applied Sciences* 12.19 (2022). ISSN: 2076-3417. DOI: 10.3390/app12199851. URL: https://www.mdpi.com/2076-3417/12/19/9851 (cit. on pp. 22, 25).

[Tam20] D. A. Tamburri. “Sustainable MLOps: Trends and Challenges”. In: *2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. Timisoara, Romania: IEEE, Sept. 2020, pp. 17–23. ISBN: 978-1-72817-628-4. DOI: 10.1109/SYNASC51798.2020.00015. URL: https://ieeexplore.ieee.org/document/9356947/ (visited on 05/20/2024) (cit. on p. 29).

[TBF+22] M. Testi, M. Ballabio, E. Frontoni, G. Iannello, S. Moccia, P. Soda, G. Vessio. “MLOps: A Taxonomy and a Methodology”. In: *IEEE Access* 10 (2022), pp. 63606–63618. DOI: 10.1109/ACCESS.2022.3181730 (cit. on p. 24).

[Tea24] T. lakeFS Team. *What is a Data Product? Guide, Definition & Examples*. 2024. URL: https://lakefs.io/blog/data-products/ (cit. on p. 19).

[Wie14] R. J. Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, p. 493. ISBN: 978-3-662-43838-1. DOI: 10.1007/978-3-662-43839-8 (cit. on pp. 31, 36, 37, 67).

[ZBC16] L. Zhu, L. Bass, G. Champlin-Scharff. “DevOps and Its Practices”. In: *IEEE Software* 33 (May 2016), pp. 32–34. DOI: 10.1109/MS.2016.81 (cit. on p. 23).

[ZYD20] Y. Zhou, Y. Yu, B. Ding. "Towards MLOps: A Case Study of ML Pipeline Platform". In: *2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE)*. Beijing, China: IEEE, Oct. 2020, pp. 494–500. ISBN: 978-1-72819-146-1. DOI: 10.1109/ICAICE51518.2020.00102. URL: https://ieeexplore.ieee.org/document/9361315/ (visited on 05/18/2024) (cit. on pp. 24, 30).

[ZZL+23] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, J.-R. Wen. *A Survey of Large Language Models*. arXiv:2303.18223 [cs]. Nov. 2023. URL: http://arxiv.org/abs/2303.18223 (visited on 07/25/2024) (cit. on p. 21).

All links were last followed on October 13, 2024.

A Requirements

Table A.1: List of requirements

ID	Requirement	Type	Priority	Rationale
DC1	Compatibility with Azure	Defined Constraint	High	The infrastructure of the team is built on Azure and while many other teams in the business unit are still on-premise, most that are already working with cloud services are using Azure as well.
DC2	Compatibility with GitHub	Defined Constraint	High	The team already uses GitHub for code repositories and GitHub Actions for CI.
DC3	No own Kubernetes for MLOps platform	Defined Constraint	High	Maintaining a dedicated Kubernetes with its complete ecosystem just to run an MLOps platform is too much effort for the team and any other team who should be able to operate this solution.
FR1	Supports classical ML, Deep Learning and LLMs	Functional	High	Although the current focus is on LLMs, future use cases are open and all possibilities must be considered and supported.
FR2	Allowed at Bosch	Functional	Medium	The tool must be allowed at Bosch, which means it has to have an appropriate license, implement the necessary security requirements and, for an external service, have passed an onboarding.
FR3	Integration with existing CI/CD concept	Functional	Medium	Since there is already a working CI/CD concept in place, it is necessary for the MLOps pipeline to integrate into the existing structures.
FR4	Supports on-premise projects	Functional	Low	Possible future use cases may not be able to run in the cloud, but only on-premises (e.g., due to high data volume or the data privacy class).
NFR1	Maintainability	Non-functional	High	Since the solution must be maintained by the team itself, the effort required for it must be low.

Continued on next page

Table A.1 – continued from previous page

ID	Requirement	Type	Priority	Rationale
NFR2	User Experience	Non-functional	High	Although the primary focus is on a team, which is technically adept and would have no problem with command line tools or configuration in code, in the long term the user base may expand to include less technically adept users from across the business unit. Therefore, a graphical user interface that is easy to understand and use is preferred.
NFR3	Reliability & Stability	Non-functional	High	Although the team is technically adept, this MLOps pipeline will be a tool for them to develop data products and should be reliable and stable in order to improve and speed up the development process. In the long run, the reliability and stability of the solution will also be important, as less technically adept users will not be able to fix problems on their own and will only use the solution if it works reliably.
NFR4	Performance & Scalability	Non-functional	Medium	While the current use cases are small enough to have no particular need for performance optimization or scalability, future business unit use cases may involve larger amounts of data and therefore require a high performance and scalable solution.
NFR5	Tools already existing at Bosch	Non-functional	Medium	From a business perspective, it is also relevant whether a particular tool is already in use at Bosch or even offered as a service. This could not only provide knowledge and experience about the tool in use, but also speed up the setup or even offer the possibility to use this tool without having to set it up and maintain it yourself.

Continued on next page

Table A.1 – continued from previous page

ID	Requirement	Type	Priority	Rationale
NFR6	Modularity & Reusability	Non-functional	Low	The team already has MLOps related tools in use. These were chosen according to the needs of the current use case. As the future use cases may have some similar and some different requirements, it is important that the solution is modular and reusable. This means that the components should not be tightly coupled, but should have clear interfaces so that they can be reused in other projects or individual components can be easily replaced by other tools.

B Rapid Review Results

Table B.1: White literature selected in Rapid Review

Title	Authors	Type	Year	Publisher
MLOps: a guide to its adoption in the context of responsible AI	Matsui and Goya	Conference Paper	2023	ACM
Machine Learning Operations (MLOps): Overview, Definition, and Architecture	Kreuzberger et al.	Journal Article	2023	
MLOps - Definitions, Tools and Challenges	Symeonidis et al.	Conference Paper	2022	IEEE
Towards MLOps: A case study of ML Pipeline Platform	Zhou et al.	Conference Paper	2020	IEEE
MLOps: A Taxonomy and a Methodology	Testi et al.	Journal Article	2022	
A Multivocal Literature Review of MLOps Tools and Features	Recupito et al.	Conference Paper	2022	IEEE
Teaching MLOps in Higher Education through Project-Based Learning	Lanubile et al.	Conference Paper	2023	IEEE
K2E: Building MLOps Environments for Governing Data and Models Catalogues while Tracking Versions	Zarate et al.	Conference Paper	2022	IEEE
MLOps Challenges in Industry 4.0	Faubel et al.	Journal Article	2023	
Infrastructure for MLOps	Sorvisto	Book Section	2023	Apress
Towards Regulatory-Compliant MLOps: Oravizio's Journey from a Machine Learning Experiment to a Deployed Certified Medical Product	Granlund et al.	Journal Article	2021	
What Is MLOps?	Alla and Adari	Book Section	2021	Apress
LLMs for Enterprise and LLMOps	Kulkarni et al.	Book Section	2023	Apress

Table B.2: Grey literature selected in Rapid Review

Title	Authors	Type	Year	Link
Choosing the Best MLOps Platform: a Comprehensive Comparison of MLOps Platforms	Valohai	Article	n.d.	https://valohai.com/mlops-platforms-compared/
MLOps Landscape in 2024: Top Tools and Platforms urlhttps://neptune.ai/blog/mlops-tools-platforms-landscape	Stephen Oladele	Article	2024	
Top End-to-End MLOps Platforms and Tools in 2024	Grig Duta	Article	2024	https://www.qwak.com/post/top-mlops-end-to-end
MLOps Services Tools and Comparison | A Quick Guide	Jagreet Kaur Gill	Article	2023	https://www.xenonstack.com/blog/mlops-tools
Compare 45+ MLOps Tools: A comprehensive vendor benchmark in '24	Cem Dilmegani	Article	2024	https://research.aimultiple.com/mlops-tools/
Open Source MLOps: Platforms, Frameworks and Tools	Nilesh Barla	Article	2024	https://neptune.ai/blog/best-open-source-mlops-tools
25 Top MLOps Tools You Need to Know in 2024	Abid Ali Awan	Article	2024	https://www.datacamp.com/blog/top-mlops-tools
Top 10 MLOps Tools to Learn in 2024	Manika Nagpal	Article	2024	https://www.projectpro.io/article/best-mlops-tools-/574
Comparing 10+ LLMOps Tools: A Comprehensive Vendor Benchmark	Cem Dilmegani	Article	2024	https://research.aimultiple.com/llmops-tools/
MLOps Architecture Guide	Stephen Oladele	Article	2023	https://neptune.ai/blog/mlops-architecture-guide
MLOps Architectural Models: An Advanced Guide to MLOps in Practice	Daniela Kolarova	Article	2024	https://dzone.com/articles/mlops-architectural-models-advanced-guide

Continued on next page

Table B.2 – continued from previous page

Title	Authors	Type	Year	Link
Machine learning operations (MLOps) framework to upscale machine learning lifecycle with Azure Machine Learning	Azure	Website	2024	https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/mlops-technical-paper
MLOps Principles	Dr. Larysa Visengeriyeva	Website	2024	https://ml-ops.org/content/mlops-principles
GUIDE TO Evaluating MLOps Platforms	Ryan Dawson	White Paper	2021	https://www.thoughtworks.com/content/dam/thoughtworks/documents/whitepaper/tw_whitepaper_guide_to_evaluating_mlops_platforms_2021.pdf
MLOps What is it How to get started	Vanenburg	White Paper	2022	https://www.vanenburg.com/wp-content/uploads/2022/05/MLOps_Whitepaper_FINAL_updated.pdf
Practitioners guide to MLOps: A framework for continuous delivery and automation of machine learning.	Khalid Salama	White Paper	2021	https://services.google.com/fh/files/misc/practitioners_guide_to_mlops_whitepaper.pdf
The Big Book of MLOps	Databricks	E Book	2022	https://blog.infocruncher.com/resources/ml-productionisation/The%20Big%20Book%20of%20MLOps%20(Databricks,%20v6,%202022).pdf

C Architecture Decision Record Format

Name	99 ADR Template					
Status	DRAFT					
Context / Problem / Issue Description						
Options considered	**Option A**	**Option B**	**Option C**	**Option D**	**Option E**	**Option F**
Name & Link						
Description						
License & Cost						
Documentation						
Infrastructure / How to run it						
Academia						
Advantages						
Disadvantages						
Requirements						
Compatibility with Azure						
Compatibility with GitHub						
No own Kubernetes needed						
Allowed at Bosch						
Supports all kinds of ML						
Supports on-premise						
User Experience						
Maintainability						
Integration with CI/CD concept						
Reliability & Stability						
Modularity & Reusability						
Performance & Scalability						
Tool already existing at Bosch						
Decision						
Justification						
Decided by						
Date of decision	<dd.mm.yyyy>					
Comments / Conditions to be met	<bullet point list, task tracking issues>					
Related Decisions	<link to related decision>					

Figure C.1: The ADR base template created for and used in this thesis.

D Identified Platforms and Tools

Platforms: AWS SageMaker, Azure ML, ClearML, Cloudera, cnvrg.io, Comet, Databricks, DataRobot, H2O.ai, IBM Watson Studio, Iguazio, Kubeflow, Metaflow, MLflow, MLRun, neptune.ai, Polyaxon, TensorFlow Extended, TrueFoundry, Valohai, Vertex AI, Weights & Biases, ZenML

***Data Labeling* Tools:** Azure ML Data Labeling, Doccano, iMerit, Labelbox, Prodigy, Scale, Segments.ai, Snorkel Flow, SuperAnnotate, Superb AI, Superwisely, V7 Darwin

***Data Versioning & Management* Tools:** Delta Lake, Dolt, DVC, LakeFS, Pachyderm, Qri, Quilt

***Feature Store* Tools:** Feast, Featureform, Hopswork, Rasgo, Tecton

***Model Training Pipeline & Workflow Orchestration* Tools:** Akira AI, Apache Airflow, Argo Workflows, BodyWork, Flyte, Kedro, Kubeflow Pipelines, Prefect

***Hyperparameter Tuning* Tools:** Hyperopt, Optuna, Ray Tune, Scikit-Optimize, SigOpt, Talos

***Experiment Tracking & Metadata Store* Tools:** Aim, CML, Guild AI, MLflow Tracking, ModelDB, TensorBoard

***Model Testing & Validation* Tools:** Alibi Explain, Arize Phoenix, Deepchecks, Fiddler, Galileo, Giskard, Great Expectations, Ragas, TruEra

***Model Registry* Tools:** AWS SageMaker Model Registry, Azure ML Model Registry, Databricks Model Registry, Kubeflow Model Registry, MLflow Model Registry, neptune.ai

***Model Deployment & Serving* Tools:** Algorithmia, BentoML, Cortex, Hugging Face, KServe (Kubeflow), LangServe, NVIDIA Triton Inference Server, OctoML, Ray Serve, Seldon Core, TensorFlow Serving, Torch Serve

***Monitoring & Observability* Tools:** Aporia, Arize AI, Evidently AI, Fiddler, Gantry, Helicone, Langfuse, Losswise, Mona, Prometheus with Grafana, Superwise.ai, Unravel Data, WhyLabs

E User Guide for the Template Repository

Initial Setup:

- Create a new repository based on the template
  - Make sure the GitHub Action is running and creates the initial resources
  - if necessary add missing Secrets or environment variables to the GitHub repository
  - if necessary approve private endpoint connections
- Configure Databricks
  - if not existing create a personal compute
  - if not existing create a PAT in Databricks
  - check out the repository in your own workspace (if needed provide a GitHub access token)
  - test creating and running a notebook
  - look into the Databricks components/functionalities
- Clone the repository locally
  - Install Python and Poetry
  - If needed add `.env` files (include for example the Databricks PAT)
  - test running the backend and frontend locally
  - optional: install the Visual Studio Code Databricks extension and configure the connection to the Databricks workspace

Development Workflow:

- create and checkout a new feature branch
- for model development:
  - either locally or on Databricks: explore data
  - either locally or on Databricks: experiment with model generation
  - log experiment runs to MLflow Tracking server
  - once ready register model to MLflow Model Registry
  - if needed manually deploy the model version to a Model Serving Endpoint in Databricks

  - productionize model training and validation code into workflow for automated execution
- for application development:
  - make changes to backend or frontend code
  - if needed switch to use manually created model endpoint of feature branch
  - test complete data product locally
  - switch to production model endpoints
- open Pull Request to DEV:
  - Get a review
- Merge into DEV: GitHub Action does:
  - Runs prepared workflow for model training, validation and logging
  - Builds Docker containers for backend and frontend
  - Deploys latest model to Model Serving Endpoint in Databricks
  - Deploys latest backend/frontend version to Azure Web App
- Test deployed data product on DEV

F Sample Use Cases

F.1 Use Case 1: Chatbot answering questions on a climate report

For sample use case 1, the components defined in Section 5.2 are covered as follows:

- *Data Labeling*: Not required.
- *Data Versioning & Management*: Covered by the integrated Delta Lake in Databricks, which is used for versioning the climate report for the RAG system.
- *Exploratory Data Analysis*: Possible via notebooks in Databricks, including rapid prototyping for developing and testing chains on feature branches.
- *Feature Store*: Available in Databricks but not needed in this application. item *Code Repository*: Provided by GitHub, with the repository additionally synced in Databricks and managed from there.
- *CI/CD*: Implemented through GitHub Actions pipelines that deploy all resources in Azure or directly in Databricks via Terraform.
- *Model Training Pipeline & Workflow Orchestration*: Not required for training or retraining an ML model but necessary for automated validation of the chain and regular updates of the vector database; covered by the workflow configuration in Databricks.
- *Hyperparameter Tuning*: Not implemented and not relevant in this application.
- *Experiment Tracking & Metadata Store*: Essential despite the absence of traditional model training; different versions of the LangChain are stored here along with validation results and metrics, such as testing, for example, against a ground truth set.
- *Model Testing & Validation*: Covered by a test step in the workflow that uses Giskard and Ragas to assess performance, response quality, and uncover potential weaknesses or risks of the model.
- *Model Registry*: Utilizes the Databricks internal MLflow Model Registry. Although the LangChain is only deployed for testing purposes, it needs to be stored in the model registry then.
- *Model Deployment & Serving*: During development, the LangChain can be served in Databricks for quick testing. For production use, it is deployed with LangServe in a FastAPI backend.
- *Monitoring & Observability*: Implemented using Langfuse. The LangServe backend sends all traces to Langfuse, which can be used for later validation of the chatbot's performance and for explainability, as it stores not only the response but also the entire chain with the retriever's context.

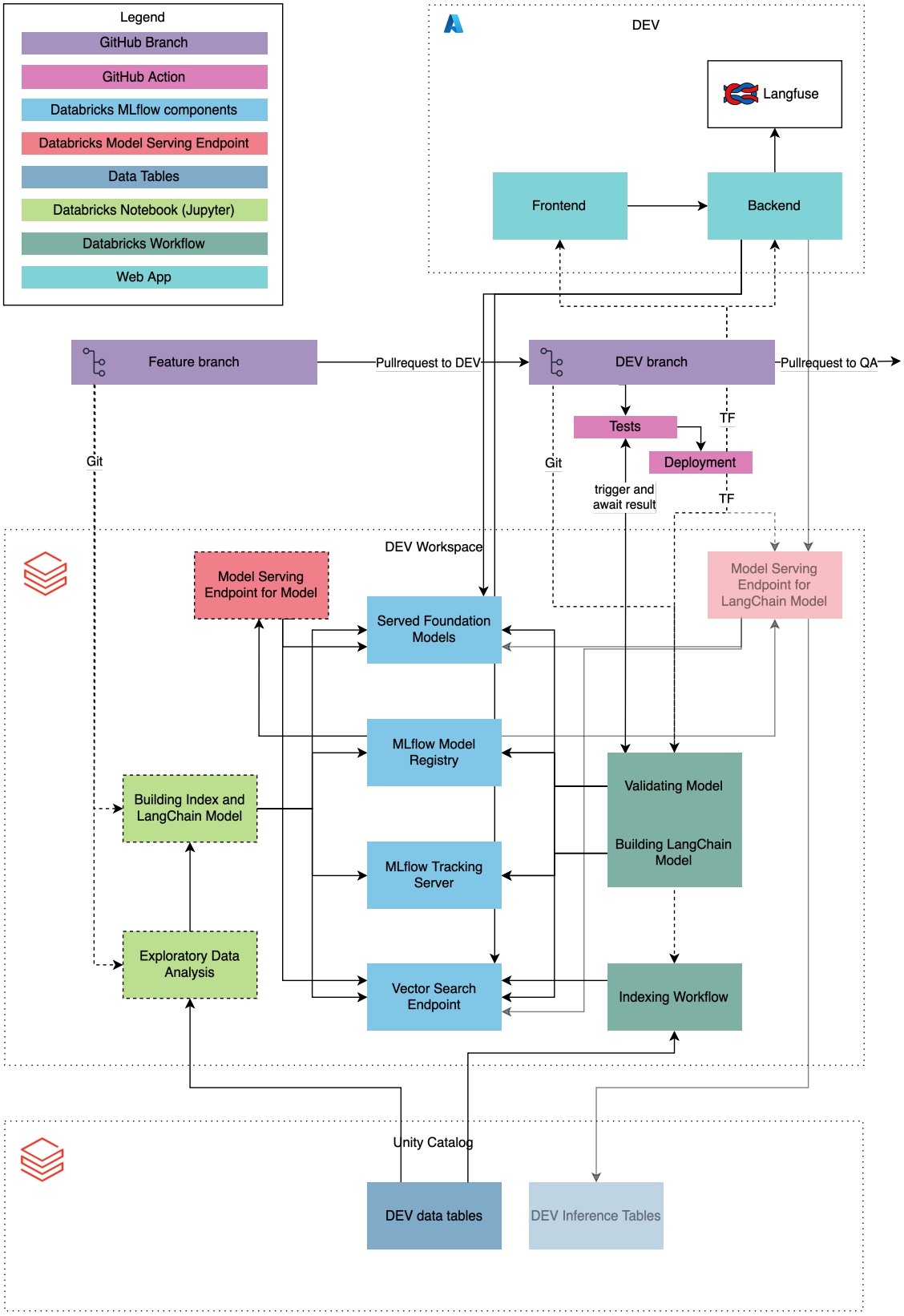


Figure F.1: MLOps workflow in DATABRICKS for sample use case 1.

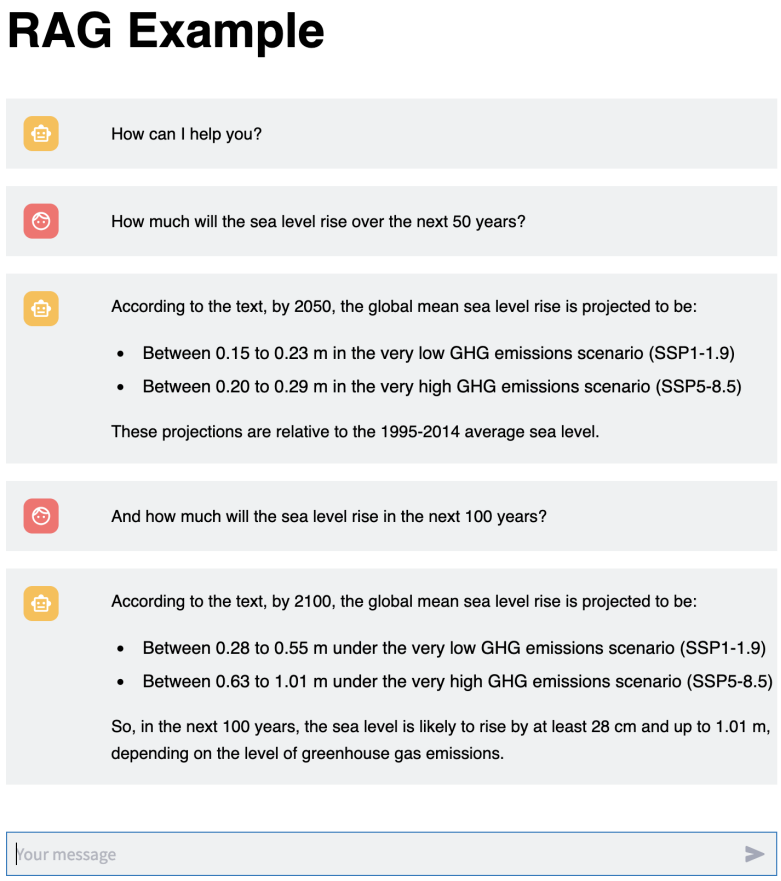


Figure F.2: Frontend of the implemented sample use case 1: Chatbot answering questions on a climate report.

F.2 Use Case 2: Classification of handwritten digits

For sample use case 2, the components defined in Section 5.2 are covered as follows:

- The *Data Labeling* component is not required, as the MNIST dataset already contains labeled data.
- The *Data Versioning & Management* component is covered by the integrated Delta Lake in Databricks, which is used for versioning and storing the MNIST data.
- *Exploratory data analysis* is possible through notebooks in Databricks and also includes initial experimentation with model development on feature branches.
- A *Feature Store* component is available in Databricks but is not needed in this example since the input consists solely of image data from which features relevant to other use cases cannot be extracted.
- The *Code Repository* component is provided by GitHub, with the repository additionally synced in Databricks and managed from there.
- For the *CI/CD* component, GitHub Actions pipelines exist that deploy corresponding resources in Azure or directly in Databricks using Terraform.
- The *Model Training Pipeline & Workflow Orchestration* component is necessary for automated training and retraining, as well as for executing test workflows, and is covered by the Databricks Workflow functionality. This integrates seamlessly with the CI pipeline, as workflows in Databricks can be triggered from a GitHub Action.
- A *Hyperparameter Tuning* component is not used in this application but could be integrated into the model training workflows to determine the best hyperparameters for the model.
- The *Experiment Tracking & Metadata Store* component is essential here to track performance on different model versions during initial development and when propagating models through the stages, making it easy to identify the best model.
- The *Model Testing & Validation* component is covered by using a test step in the workflow that checks the performance of the model on a test set and utilizes Giskard to examine the model for potential issues.
- The Databricks internal MLflow Model Registry is used as the *Model Registry* to store the developed PyTorch classification model.
- In regard to *Model Deployment & Serving*, the model is served in Databricks and made available via an endpoint secured with a token.
- For the *Monitoring & Observability* of the model, Databricks' built-in capability to store requests is used, which could be reviewed or displayed in dashboards. Additionally, based on this data, validation of the model's performance can be conducted, or potentially a retraining initiated.

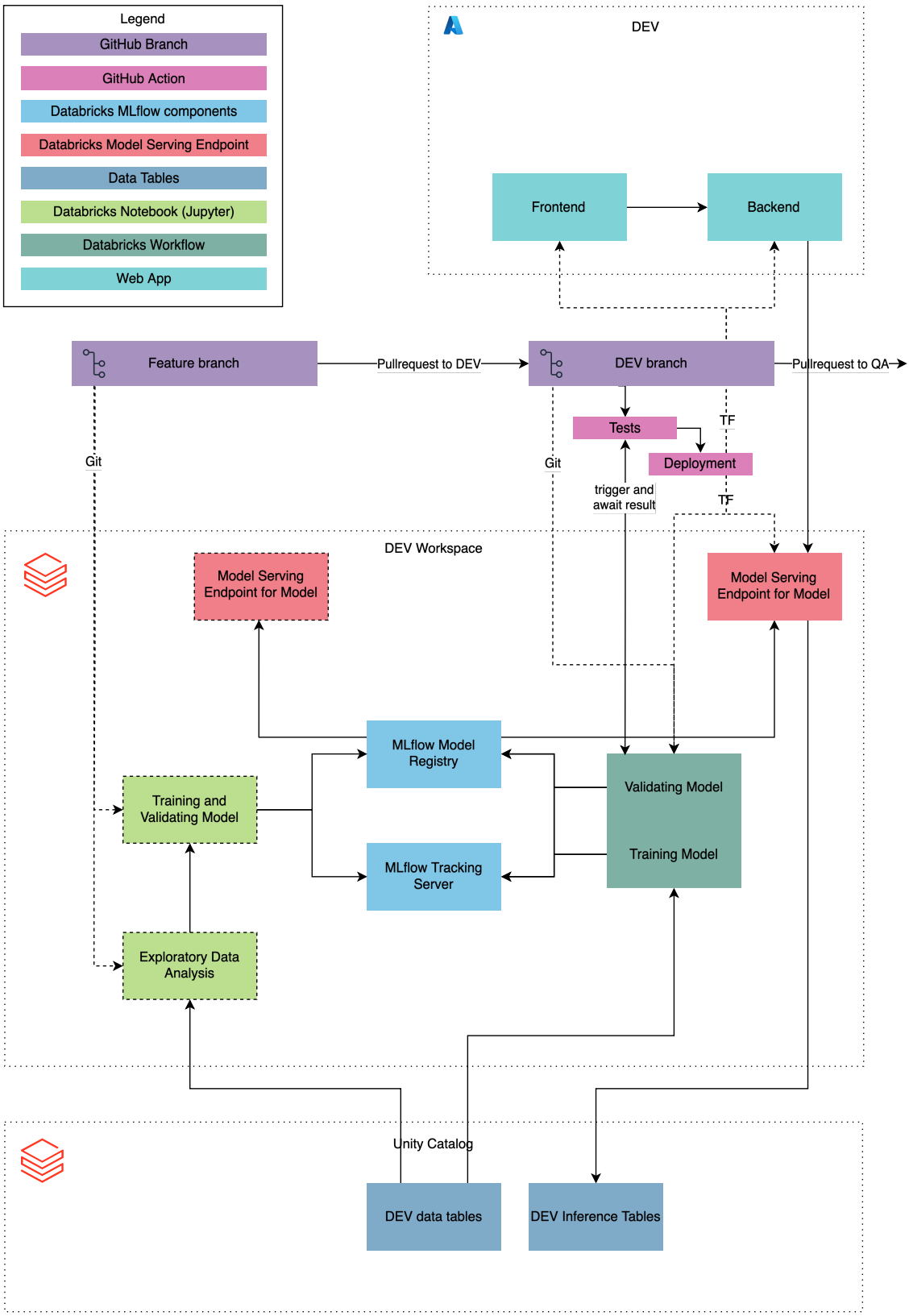


Figure F.3: MLOps workflow in DATABRICKS for sample use case 2.

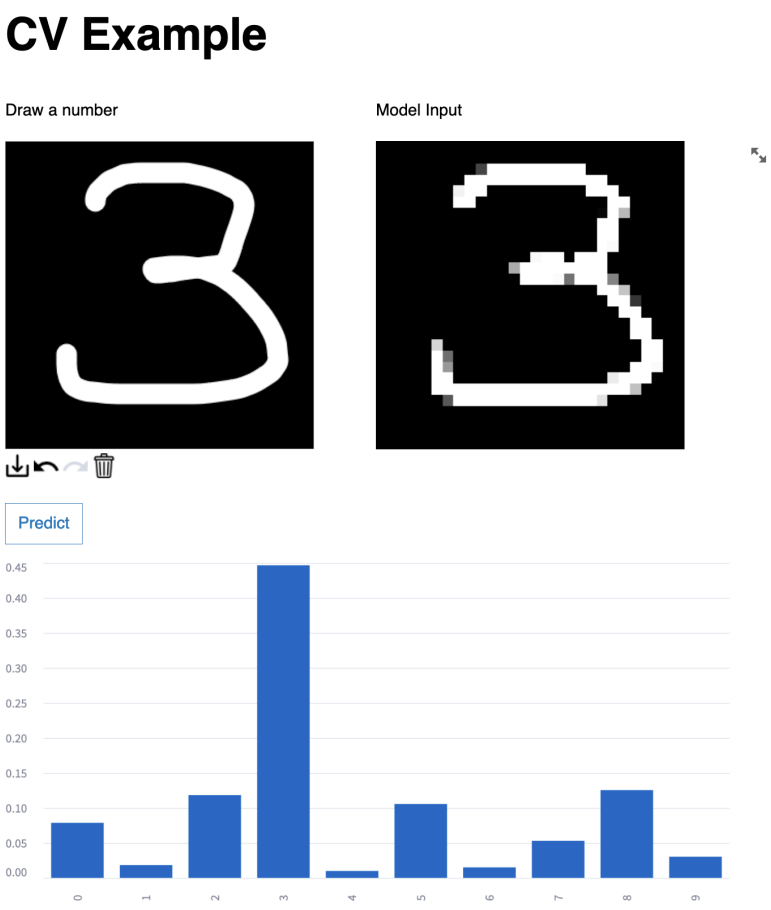


Figure F.4: Frontend of the implemented sample use case 2: Classification of handwritten digits.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Druck-Exemplaren überein.

Datum und Unterschrift:

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted hard copies.

Date and Signature: