



UNIVERSITÉ DE NAMUR
Faculté d'informatique
Année académique 2024-2025

**Automating ML Pipelines: A
MLOps-GitOps Workflow
implementation**

Benjamin Jonard

..... (Signature pour approbation du dépôt - REE art. 40)

Promoteur : Benoit Frenay

Co-promoteur : Jerome Fink

Mémoire présenté en vue de l'obtention du grade de Master 60 en Sciences Informatiques

Faculté d'Informatique – Université de Namur

RUE GRANDGAGNAGE, 21 ● B-5000 NAMUR(BELGIUM)

Abstract

Machine learning projects face significant deployment challenges, with only a small percentage successfully reaching production due to operational complexity. This thesis addresses these challenges by developing and implementing a state-of-the-art MLOps-GitOps workflow designed to support machine learning projects from initiation through production maturity.

Following a comprehensive literature review of current MLOps methodologies and frameworks, we implement a modular workflow using a two-phase approach: initial validation with a controlled demonstration model, followed by a design tailored for integration with the real-world LSFB (Langue des Signes de Belgique Francophone) production system.

The implementation integrates enterprise-level technologies including Kubernetes, Docker, Helm, Airflow, and KubeFlow pipelines, applying GitOps methodology consistently across DataOps and MLOps processes. The resulting workflow successfully demonstrates how a state-of-the-art MLOps-GitOps approach can bootstrap machine learning projects and support their progression toward greater maturity, providing a reusable foundation for diverse data engineering and machine learning projects across various organizational scales.

Keywords— MLOps, DevOps, DataOps, GitOps, Machine Learning Deployment, CI/CD, Pipeline Automation, KubeFlow, Airflow, Python, Kubernetes

Contents

1	State of the art: Machine learning development and operations workflow (MLOps)	9
1.1	Introduction	9
1.2	Definitions	9
1.2.1	Containerization	9
1.2.2	Kubernetes	10
1.2.3	Helm	10
1.2.4	DevOps	10
1.2.5	GitOps	11
1.2.6	CI/CD Workflows	11
1.2.7	S3-Storage	12
1.3	Methodology	12
1.3.1	Literature review (LR)	12
1.3.2	Research questions	12
1.3.3	Article Relevance and Inclusion/Exclusion Criteria	13
1.4	DataOps in MLOps Workflows	13
1.4.1	DataOps Definition and Principles	13
1.4.2	DataOps Workflow	13
1.5	MLOps	14
1.5.1	Description	15
1.5.2	Challenges	16
1.5.3	Roles in MLOps	16
1.5.4	Workflow	17
1.5.5	Architecture, infrastructure and existing platforms	24
1.5.6	DataOps Integration with MLOps	26
1.5.7	Maturity Models	27
1.5.8	GitOps in MLOps workflow	27
1.6	Tools and Platforms	29
1.6.1	Version Control and CI/CD Pipelines	29
1.6.2	Apache Airflow	29
1.6.3	Kubeflow	29
1.6.4	Databricks	30
1.6.5	Argo	30

1.6.6	MLflow	30
1.6.7	ZenML	31
1.6.8	TensorFlow Extended	31
1.6.9	Dagster	31
1.7	Results	31
1.8	Conclusion	32
2	Thesis project: Implementing a MLOps/GitOps workflow	35
2.1	Introduction	35
2.2	Use Case Definitions	36
2.2.1	Collaborative Model Development	36
2.2.2	Data Storage and Analysis	36
2.2.3	Model Versioning and Storage	38
2.2.4	Model and Tool Deployment	38
2.2.5	Workflow Automation	38
2.3	Tools Selection Rationale	38
2.3.1	Python	38
2.3.2	Git Repositories and CI/CD	38
2.3.3	IAC, Containers and Registry	39
2.3.4	Kubernetes	39
2.3.5	Airflow	39
2.3.6	Kubeflow	39
2.3.7	ArgoCD	39
2.4	Infrastructure and Workflow	40
2.4.1	General workflow	40
2.4.2	Git Repositories	42
2.4.3	GitOps	44
2.4.4	DataOps pipelines with Airflow	46
2.4.5	MLOps pipelines with Kubeflow	46
2.4.6	Storages	46
2.4.7	Model development	47
2.5	Roles within our MLOps workflow	48
2.5.1	DevOps Engineers	49
2.5.2	Data Engineers	49
2.5.3	MLOps Engineers	49
2.5.4	Model Engineers	49
2.5.5	Software Engineers	49
2.5.6	Example of minimal list of git repositories per roles/teams within an Organisation	50
2.6	Workflow pipelines	50
2.6.1	DevOps CI/CD pipelines	50
2.6.2	DataOps pipelines	51
2.6.3	MLOps pipelines	53
2.7	Limitations and Critical Assessment	55

<i>CONTENTS</i>	5
2.8 Future Work	56
2.9 Conclusion	56
Bibliography	59

Prologue

Introduction

Software development has changed a lot over the years, moving from manual steps to automated workflows. Old methods had long manual separate steps, often causing problems. To deliver faster and improve quality modern development has shifted to automation. Machine Learning development also moves towards more automation[15].

According to[26], only a small percentage of machine learning projects successfully reach production deployment, due to issues and operational complexity[15]. The emergence of MLOps as a distinct discipline represents a response to these challenges, offering structured approaches to model development, deployment, monitoring, and governance[40]. This thesis begins with a comprehensive state-of-the-art review of MLOps methodologies, frameworks, and tools currently employed across industry and research domains. Our review is motivated by the need to establish a solid theoretical and practical foundation for developing a flexible MLOps workflow that can address the diverse requirements of modern enterprises and research organizations. We aim to identify in the literature the core components and principles that constitute effective MLOps implementations[24].

The insights gained from this review directly inform our subsequent implementation project, where we develop a modular, adaptable MLOps workflow designed to accommodate various scales of machine learning projects. Our implementation strategy follows a two-phase approach: first, we develop and validate our workflow using a controlled demonstration model that shows core functionality in a simplified context. This allows us to establish baseline capabilities and verify fundamental workflow mechanics without the complexities of a production system. Following this initial validation, we demonstrate the workflow's flexibility and robustness by applying it to a real-world production system—the LSFB (Langue des Signes de Belgique Francophone) model developed at our university[11]. By adapting our workflow to enhance this existing system, we provide tangible evidence of its practical utility and adaptability.

Acknowledgements

I would like to express my sincere gratitude to all those who contributed to the successful completion of this thesis. First and foremost, I extend my deepest appreciation to my thesis promoters Benoit Frenay and Jérôme Fink for their invaluable guidance, continuous support, and expertise throughout this research project. Their insights into Machine Learning Development and practical feedback were

instrumental in shaping the direction and quality of this work. I am grateful to the team behind the LSFB (Langue des Signes de Belgique Francophone) project for providing access to their dataset. Their collaboration was essential in demonstrating the real-world applicability of our MLOps-GitOps workflow, and their domain expertise greatly enriched the practical validation of our implementation. Special thanks to my fellow students and peers who provided valuable feedback during the development process, particularly regarding the integration of GitOps methodology with our MLOps workflow. Their suggestions and collaborative discussions helped refine our approach and improve the overall solution. Finally, I am grateful to my family and friends for their support throughout my master's program. This research would not have been possible without them.

Chapter 1

State of the art: Machine learning development and operations workflow (MLOps)

1.1 Introduction

This chapter lays the groundwork for our implementation by exploring the current landscape of MLOps practices, identifying key challenges in machine learning development, and criteria for implementing an effective MLOps workflows. Through this review, we aim to bridge theoretical understanding with practical application, ultimately contributing to more reliable, efficient, and maintainable machine learning systems in production environments.

1.2 Definitions

Having established the context and objectives of our research, we will now define the fundamental concepts that underpin MLOps. By establishing this common vocabulary, we aim to make our analysis and subsequent implementation accessible to a wider technical audience. It requires only a prior general knowledge about machine learning, software development, or data engineering.

1.2.1 Containerization

Containerization is a technology that packages applications and their dependencies into isolated, lightweight containers, ensuring consistent and portable execution across various environments. This approach eliminates common problems by providing a standardized runtime environment[9].

In the context of MLOps, containerization is crucial for the reliable and reproducible deployment of machine learning models. By encapsulating models along with their software dependencies and environment configurations, containers prevent version conflicts and isolate resource usage. This isolation

ensures that different models or services can run simultaneously without interfering with one another, enabling scalable and maintainable ML systems.[40](p.80)

While Docker[9] is the most widely adopted containerization platform, other tools such as Podman, containerd, and CRI-O are also gaining traction, each with their own advantages in terms of security, rootless execution, and integration with orchestration systems like Kubernetes.

1.2.2 Kubernetes

Kubernetes[21] is a container orchestration platform that automates the deployment, scaling, and management of containerized applications. It enables infrastructure as code through tools like Helm, simplifying application deployment and configuration. By using features such as namespaces, resource quotas, and specialized node types (e.g., GPU nodes for machine learning), Kubernetes allows multiple teams to securely share resources while maintaining isolation. Its portability ensures compatibility across various Kubernetes providers, making it a flexible and scalable solution for diverse projects.

Docker with Kubernetes together simplifies deployment strategies and enables scalable hosting of ML models, though it lacks native ML performance management capabilities on its own[40](p.81).

1.2.3 Helm

is a package manager for Kubernetes that simplifies the deployment and management of applications[39]. It's used to define the Infrastructure-as-code as charts with templates of Kubernetes manifest.

1.2.4 DevOps

DevOps is a set of practices that combines software development (Dev) and IT operations (Ops) to shorten the development lifecycle while delivering high-quality software continuously[41]. It emphasizes automation, continuous integration/continuous deployment (CI/CD), and collaboration between teams.

The DevOps workflow typically includes[41]:

- Plan (requirements gathering),
- Code (development),
- Build (compilation/packaging),
- Test (validation),
- Release (versioning),
- Deploy (infrastructure provisioning),
- Operate (runtime management),
- Monitor (performance tracking).

We will see later that MLOps extends DevOps principles but introduces additional complexities due to the experimentation, data dependencies, and model retraining needs[26].

DevSecOps integrates security practices into DevOps, ensuring secure software delivery.

1.2.5 GitOps

GitOps[32] is a DevOps methodology that uses Git repositories as the single source of truth for infrastructure and application code and configurations[12]. GitOps include[32]:

- Declarative infrastructure (defined as code, e.g., Kubernetes manifests),
- Automated synchronization (tools like ArgoCD apply Git changes to clusters),
- Version control and auditability (changes are tracked via Git history).

Unlike traditional CI/CD (push-based deployments), GitOps follows a pull-based model, where the cluster periodically checks for updates in Git and reconciles its state accordingly[18]. According to[12] *Pull-based approaches should be chosen whenever possible because they are deemed more secure and better practices for implementing GitOps*

1.2.6 CI/CD Workflows

CI/CD (Continuous Integration/Continuous Delivery or Deployment) automates software delivery pipelines. Common branching strategies include:

GitHub Flow

A lightweight workflow where[34]:

- The main branch is always deployable,
- Feature branches are merged via pull requests (PRs),
- Changes are deployed immediately after merging.

GitFlow

A structured workflow with long-lived branches[34]:

- main (production-ready code),
- develop (integration branch),
- Feature/hotfix/release branches.

Trunk-Based Development

- Developers commit directly to a single branch (main/trunk)[36]

1.2.7 S3-Storage

Amazon S3 (Simple Storage Service) is a cloud-based object storage service that provides scalable, durable, and highly available storage infrastructure. In the context of MLOps[26], S3 serves as a critical component for storing machine learning models, datasets, and artifacts throughout the ML lifecycle. S3 provides virtually unlimited storage capacity with high durability for large ML models and datasets. It allows teams to maintain multiple versions of models for rollback and testing. S3 integrates seamlessly with popular ML platforms like Kubeflow and MLflow, offering multiple storage classes to optimize costs[26].

1.3 Methodology

In this section we explain our methodology and research question.

1.3.1 Literature review (LR)

We conducted a Literature Review in order to refine our research questions and find the key concepts and gaps in MLOps Workflows. We proceeded as follows:

1. Define our research questions
2. Define articles inclusion and exclusion criteria
3. Refine our set of keywords
4. Querying publications search engines with our keywords
5. Forward tracking and Back tracking

We used this method because it is the *de facto* approach to control biases and draws conclusions that stay partial due to a non-complete coverage of the existing literature.

1.3.2 Research questions

We first defined several research questions and then refined the one that we deemed the most relevant. Our first set of questions were:

- What are the particularities of an MLOps workflow compared to a classic DevOps CI/CD workflow?
- What are the new requirements and challenges in MLOps?
- Are there/What are already mature tools to help development teams create pipelines for their ML projects
- What limitations of those tools can be identified?

Our final research question is "*How can a state-of-the-art MLOps-GitOps workflow be implemented or proposed to initiate a machine learning project and support its progression toward greater maturity?*"

1.3.3 Article Relevance and Inclusion/Exclusion Criteria

The selection of articles was guided by the following criteria:

- Focus on MLOps, with a preference for Python-based frameworks and architectures/products deployed on Kubernetes.
- Exclude articles about using Machine Learning or AI for DevOps (the reverse of MLOps).
- Exclude articles that do not provide additional insights beyond those already included.
- Discard articles centered on hardware-specific considerations, as Kubernetes abstracts away most hardware concerns.
- Retain only a few documentation sources related to hardware, specifically for understanding node requirements.
- Exclude cloud-based or SaaS-focused solutions.
- Include some documentation on related practices such as DevOps, GitOps, and DataOps to support broader definitions and architectural context.

1.4 DataOps in MLOps Workflows

Machine Learning Operations (MLOps) workflows heavily depend on reliable, high-quality data pipelines[35]. This is where DataOps comes into play as a critical component of any comprehensive MLOps strategy. DataOps applies DevOps principles to data management, ensuring data important role in ML systems.

1.4.1 DataOps Definition and Principles

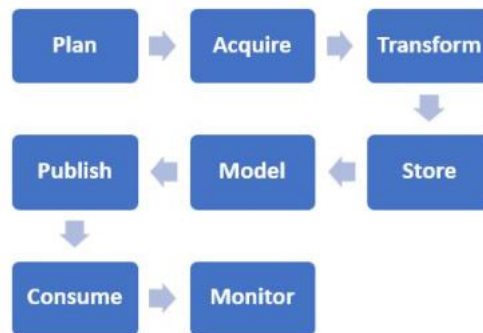
DataOps is a methodology that applies agile and DevOps principles to data management, fostering better collaboration, automated workflows, and seamless integration between data engineers and data users[31]. It represents the alignment of people, process, and technology to enable the rapid, automated, and secure management of data. The core principles of DataOps center around several key practices. Automation plays a critical role by streamlining the creation, testing, and deployment of data pipelines. Quality is maintained through continuous monitoring and validation of data to ensure reliability and accuracy. Observability provides transparency into data pipeline operations, enabling teams to detect and resolve issues quickly. Version control is essential for tracking changes in data, schemas, and transformations, ensuring reproducibility and accountability. Finally, the use of an agile methodology brings iterative, collaborative development practices to data workflows, allowing teams to adapt quickly to changing requirements.

1.4.2 DataOps Workflow

The DataOps framework integrates methodologies from Agile, DevOps, data management and analytics.

Below are the key steps derived from a systematic literature review[10]:

Figure 1.1: DataOps lifecycle[10]



Data Orchestration

Data orchestration involves automating the flow of data from its source through various transformation steps to its final destination. This helps reduce human error and makes the process more efficient. It's also important to carefully manage how data moves from one stage to the next—for example, from raw data to cleaned data and then to analysis-ready formats—so that everything flows in the correct order.

Data Governance

Good data governance means making sure the data is accurate and trustworthy. This includes checking and cleaning the data, so it's consistent and correct. It also means protecting the data through encryption and access controls, and ensuring that all processes follow legal regulations such as GDPR.

Continuous Integration and Deployment (CI/CD)

CI/CD practices help teams work together smoothly by keeping track of changes in both code and data. They also involve automatically testing pipelines to catch problems early, which helps maintain quality and reliability as the system evolves.

Monitoring and Observability

Monitoring is about checking how the system is performing—measuring things like speed, error rates, and resource usage. Observability adds transparency by logging changes and tracking how data is accessed, which helps with debugging and maintaining trust in the system.

1.5 MLOps

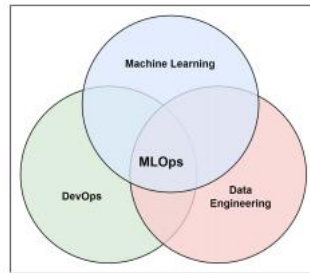
In this section, we provide an overview of MLOps as presented in current literature. We begin by introducing foundational concepts and defining the roles involved in MLOps, followed by a discussion of typical workflows and pipeline architectures.

1.5.1 Description

MLOps or Machine Learning Operations is the organisation and automation of Machine learning development workflow and operations. It's an extension of DevOps practices adapted to machine learning model development and data operations[19].

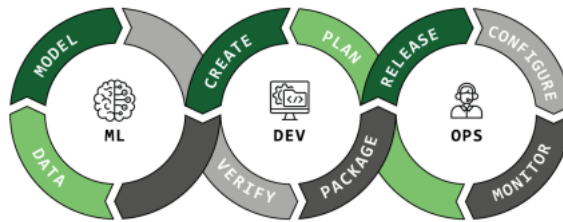
MLOps is build upon DevOps, Data Engineering and Machine Learning as stated by[39].

Figure 1.2: MLOps field[39]



MLOps with clearly defined workflow and infrastructure can provide transparency and accountability to Upper Management and Business.[40](p.11)

Figure 1.3: MLOps lifecycle[38]



MLOps is essential for data teams and organisations, as it involves a complex integration of technologies, processes, and people, requiring ongoing discipline and time to implement effectively. It is not a one-time task but a continuous effort to ensure successful machine learning operations[40](p.38).

A survey in 2021[24] already show that Machine Learning development was evolving towards team-based development. They also show that in a result the adoption of DevOps practices is increasing and the required skills in infrastructure and deployment are evolving.

In this regard, they also identified the following obstacles among others[24]:

- Accessibility of data
- Building training pipelines
- Deployment and target environment (on-premise, cloud)
- Tracking and comparing training

- Collaborating on projects
- Lack of guidelines

Since then, we will see that some of those difficulties have been tackled, tools exist and are currently reviewed in the literature.

In 2025, in their systematic literature review[42], they still consider MLOps as an emerging field. But they admit, publications are still limited. Variations in defining and grouping steps for creating and maintaining models arise from differing interpretations, model architectures, and usage methods, leading to diverse implementation approaches.

1.5.2 Challenges

Adopting MLOps presents challenges, as highlighted by[19], which categorizes them into organizational, ML system, and operational issues. Organizational challenges include shifting to a product-oriented mindset, addressing skill shortages, and fostering multidisciplinary teamwork, while ML system challenges focus on scalability and fluctuating demand, and operational challenges emphasize automation, governance, and reproducibility. These challenges are closely tied to the deployment stage in ML workflows, which involves critical steps like data management, model training, and deployment, each accompanied by issues such as data dispersion, resource constraints, and also ethical concerns[26]. All together, these challenges and considerations mark the complexity of implementing MLOps effectively.

In their publication, a Microsoft team[3] outlined challenges for MLOps implementation, including Data Availability, Collection, Cleaning, and Management, Education and Training, and Hardware Resources. They also highlighted the need for End-to-end Pipeline Support, Model Evolution, Evaluation, and Deployment, and effective Collaboration and Working Culture. Additionally, challenges like Integrating AI into Larger Systems, AI Tools, and Scale were emphasized, along with the importance of Guidance and Mentoring for skill development. These challenges highlight the multifaceted nature of MLOps, requiring both technical and organizational solutions.

1.5.3 Roles in MLOps

Machine learning development has evolved into a collaborative, team-based process involving diverse roles. Each role, from business experts to technical architects, contributes to the lifecycle of Machine Learning models. Their responsibilities include ensuring model maintenance, optimization, trust in deployment, and mitigating business risks[40](p.22).

The data scientist is often considered the central role in this process. However, within the context of MLOps, the literature identifies several key roles that collaborate to support the development and deployment of machine learning models. These roles include[40](p.14): Subject-Matter Experts, Data Scientists, Data Engineers, DevOps engineers, Software engineers, Model Risk Managers, Machine Learning architects. Additional literature further refines these roles, proposing the following[19]:

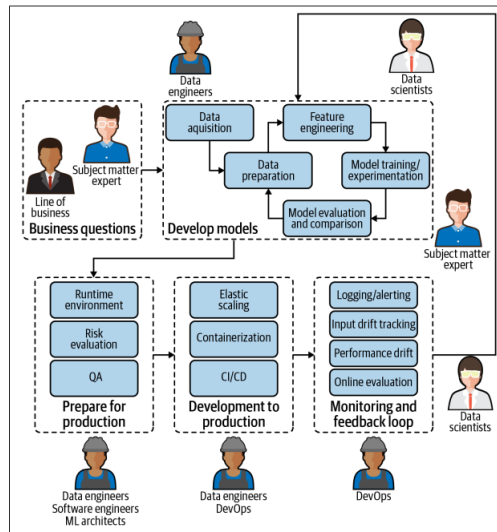
- Business Stakeholder
- Solution Architect

- Data Scientist
- Data Engineer
- Software Engineer
- DevOps Engineer
- ML Engineer/MLOps Engineer

Each person—from the subject-matter-expert on the business side to the most technical machine learning architect—plays a critical part in the maintenance of ML models in production[40](p.22).

In figure 1.4, they propose a picture of the model lifecycle and roles within an organisation. Their model is divided into five parts: Business Questions, Develop Models, Prepare for Production, Deploy to Production, and Monitoring and Feedback. Roles are assigned to each part; however, the terminology MLOps Engineer is not used, despite its prevalence in the literature[19]. The role of an MLOps Engineer or Architect would typically involve defining the overall workflow and ensuring its coherence and scalability.

Figure 1.4: Machine learning model life cycle inside an average organisation[40](p.6)

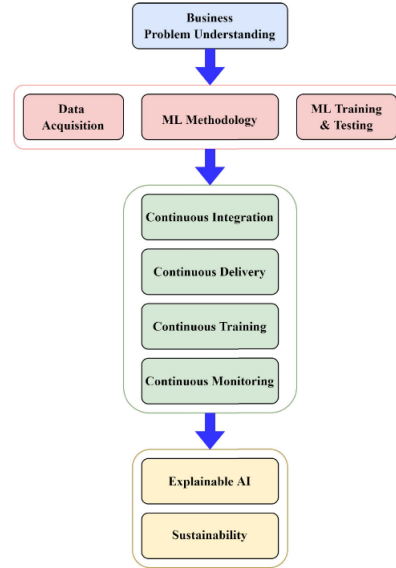


1.5.4 Workflow

Establishing an MLOps workflow is beneficial for organisations. This section outlines what constitutes an effective MLOps workflow according to current literature and best practices.

The literature presents several complementary perspectives on MLOps workflows[40, 13, 19]. In their taxonomic study[39], the authors propose a general workflow that emphasizes the initiation process flowing from business requirements into an automated MLOps workflow. This high-level view demonstrates how business needs drive technical implementation decisions 1.5.

Figure 1.5: Proposed workflow in[39]



The more complete proposition of an MLOps workflow was found in[19]. It split the workflow in 4 phases as shown in figure 1.6.:

- MLOps project initialization that involves analyzing the business problem, designing the system architecture and selecting technologies, deriving the ML task, identifying required data, and connecting to raw data for initial analysis.
- Project initiation with the definition of requirements and implementation of DataOps pipelines
- Experimentation were development of the model occurs
- Automated MLOps pipelines to prepare for production

Figure 1.7 show a simple MLOps workflow representation that also integrate DataOps.

Good practices and essential components

In the review[19], they extracted good practices and principles to implement a MLOps workflow. We complete their list by adding the State-of-the-art MLOps component that should be used for any models as described in [7].

- CI/CD automation
- Pipeline/Workflow orchestration
- Reproducibility using pipeline orchestrator

Figure 1.6: MLOps Architecture proposed in [19]

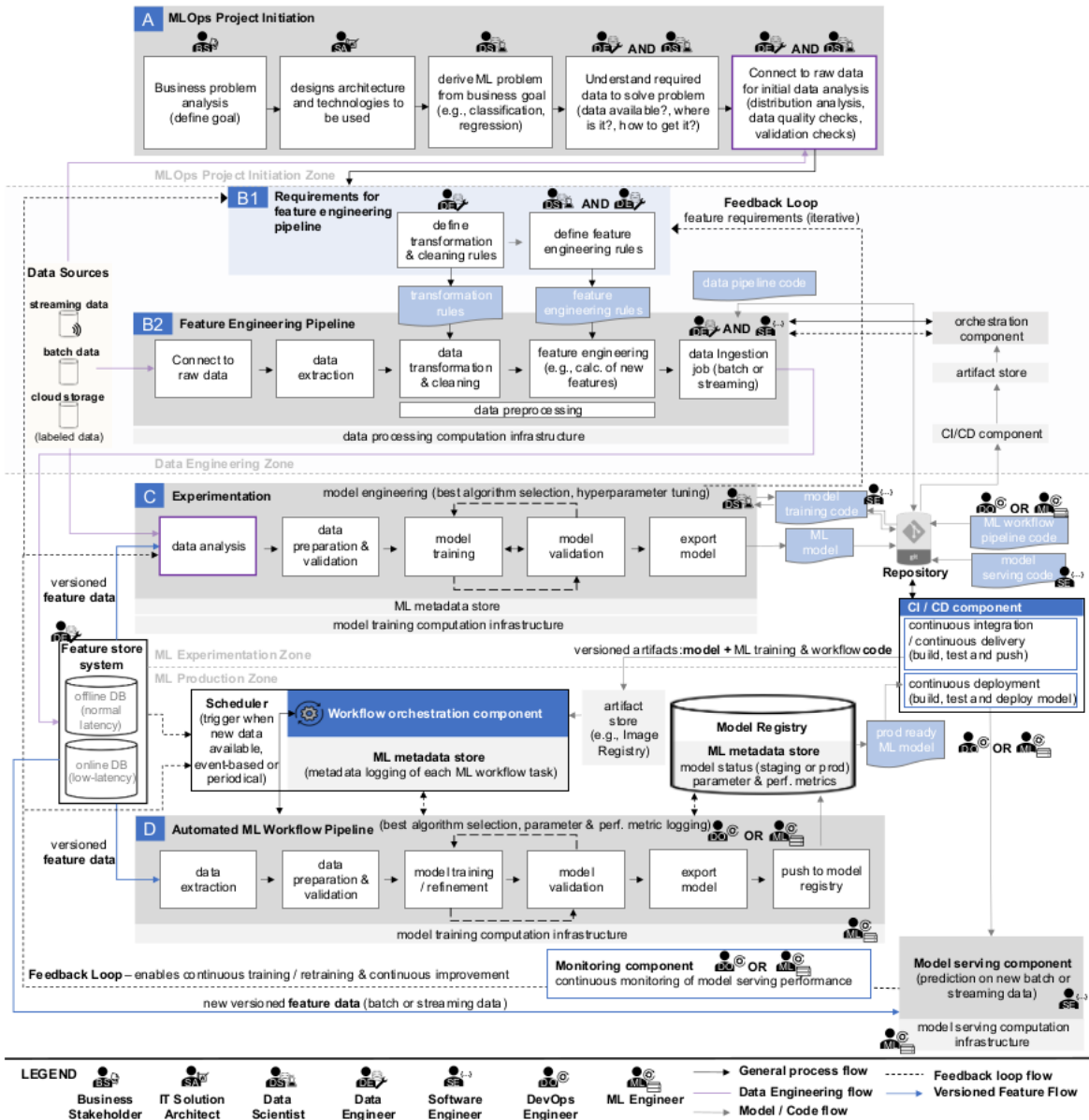
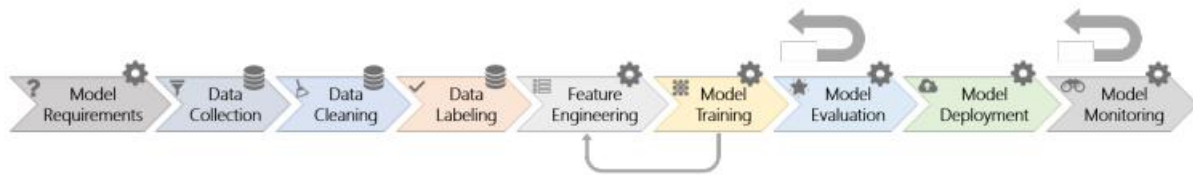


Figure 4. End-to-end MLOps architecture and workflow with functional components and roles

Figure 1.7: MLOps workflow[3]



- Versioning of data, code and model with model registry, code repositories and container registries
- Collaboration with collaborative development platform (GitHub, Gitlab, Azure Devops)
- Continuous ML training and evaluation with well-defined triggers for pipelines
- ML metadata tracking
- Continuous Monitoring (Monitoring system)
- Feedback loops

As we explained earlier those principles shares a lot with actual DevOps practices with the addition of ML considerations like training and metadata tracking.

Model development workflow

The model development process follows CI/CD principles adapted for machine learning contexts. When data scientists push code, metadata, and documentation to central repositories, they trigger automated pipelines that ensure quality and consistency[42]. An example pipeline includes the following steps[40](p.74).

Develop the model:

- Create model artifacts
- Store artifacts in long-term storage
- Run basic checks
- Generate fairness and explainability reports

Deploy to a Test Environment:

- Validate ML and computational performance
- Conduct manual validation

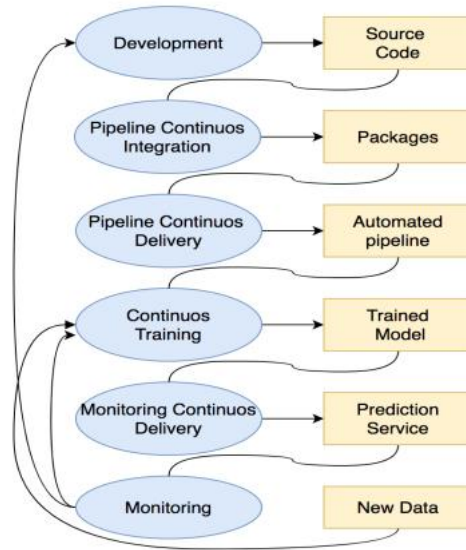
Deploy to Production:

- Roll out the model incrementally (e.g., as a canary deployment)

- Fully deploy the model once validated

Figure 1.8 presents a MLOps pipeline derived from[12]. It illustrates how each step generates artifacts that are passed to the next stage. The figure also highlights that new data and monitoring are triggers for continuous training and ongoing development.

Figure 1.8: MLOps pipeline[12]



Training Consideration

Model training involves feeding the selected model with a dataset to learn data patterns. The model training infrastructure provides computational resources (CPUs, RAM, GPUs) in either distributed or non-distributed configurations, with scalable distributed systems. It's usually implemented through local machines, cloud computing, or worker nodes using frameworks like Kubernetes[19, 15].

Model training primarily occurs in secure on-premises environments, accessing data from connected lakes[15]. Due to strict security policies, some isolated systems prohibit internet connectivity to external cloud services. It requires practitioners to use pre-approved tools from internal repositories, but it ensures data protection for sensitive information[15].

Environmental impact is a concern, as ML training increases energy consumption and CO₂ emissions[26]. Privacy is also critical, with risks like membership inference attacks[26, 22]. A major concern is the economic cost due to computational resources, particularly in Natural Language Processing[26].

Online training doesn't follow the same path as regular offline training that is usually performed in an offline training environment. It's recommended to deploy a shadow copy of the model in production for online training[40]. We will not explore further this type of training in this paper.

Hyper-parameter Optimization (HPO) HPO selects optimal hyper-parameters (e.g., decision tree depth, neural network layers) and often requires multiple training cycles, making it computationally

expensive. Large datasets further complicate HPO by increasing training time per search[26]. Defining a complete search space is often impractical due to insufficient problem knowledge, limiting state-of-the-art HPO adoption.

Testing and Validation

Model verification ensures generalization, robustness, and compliance with functional requirements. This process involves three key steps: requirement encoding, formal verification, and test-based verification[26].

Beyond accuracy, factors like robustness, security, transparency, fairness, and safety are critical for building reliable and ethical AI[22]. Defining appropriate evaluation metrics requires close collaboration between technical teams and business stakeholders[26]. Metrics must align with actual business objectives to ensure model value. This alignment often requires cross-disciplinary collaboration to establish meaningful success criteria.

Deploy in production

Maintaining machine learning systems in production presents unique challenges that differ from traditional software systems. While DevOps principles provide useful foundations for system maintenance, ML systems require specialized approaches due to their distinct characteristics[26].

The survey [26] review key challenges which include the lack of standardized methods for collecting telemetry data and difficulties in obtaining accurate labels during operation.

These differences have led to the development of specific practices for production ML systems that build upon but extend beyond conventional DevOps methodologies[13, 26].

Deployment Strategies govern how software (or ML models) are rolled out in production. Key concepts include [9, p.77]:

- Integration: Merging code changes into a shared repository and running automated tests.
- Delivery: Packaging and validating a model for production (e.g., containerization, artifact storage).
- Deployment: Releasing the model to target infrastructure (manual or automated).
- Continuous Delivery (CD): Ensuring the model is always production-ready (even if deployed manually).
- Continuous Deployment (CD): Fully automated releases upon passing tests.
- Release: Decoupling deployment from serving traffic (e.g., A/B testing, shadow deployments).

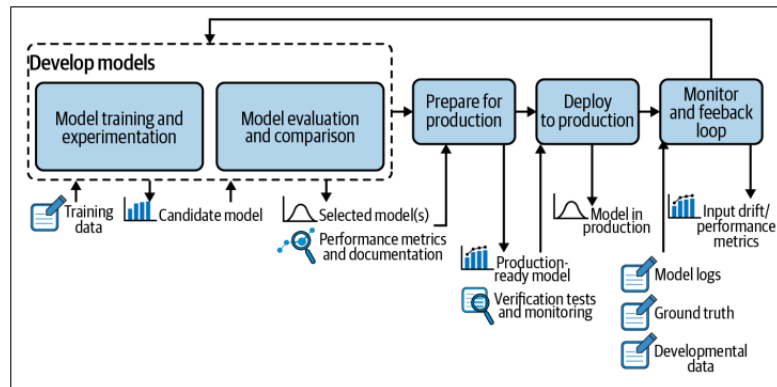
Common ML deployment strategies[13]:

- Blue-Green Deployment: Two identical environments; traffic switches post-validation.
- Canary Release: Gradual rollout to a subset of users.
- Shadow Mode: New model runs alongside the old one without affecting traffic.

Monitor and FeedBack

Understanding Model Degradation Once a machine learning model is trained and deployed in production, there are two approaches to monitor its performance degradation: ground truth evaluation and input drift detection. Understanding the theory behind and limitations of these approaches is critical to determining the best strategy.[40](p.89)

Figure 1.9: Continuous delivery for end-to-end machine learning process[40](p.95)



As explained in[40](p.103), Traditional software is designed to meet fixed specifications and maintains its functionality after deployment. In contrast, ML models are defined by their statistical performance on specific data, making them prone to performance degradation as data properties change. Beyond typical software maintenance, ML systems require careful monitoring of performance drift, with ground truth-based evaluation being essential and drift detection serving as an early warning. Retraining on new data is the primary solution, though model adjustments are also possible. Improved models can be validated using shadow scoring or A/B testing to ensure enhanced performance before deployment.

The frequency of model retraining depends on factors like the domain, the cost of retraining, and the potential performance improvement. Teams must balance these considerations, such as whether the retraining cost justifies the performance gain or if sufficient new data is available to enhance the model.[40](p.86)

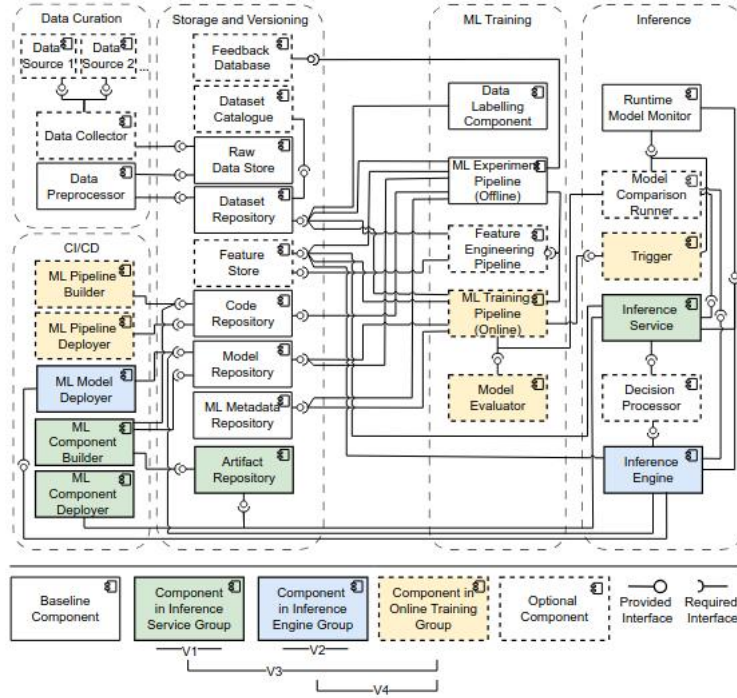
AutoML

AutoML automates the machine learning process, including tasks like data preparation, model creation, hyperparameter tuning, and evaluation, making it simpler and more accessible. It is increasingly integrated into MLOps platforms, enabling companies to deploy ML models into production more efficiently. Major cloud platforms now offer AutoML tools, enhancing productivity and feasibility for cloud-based ML projects. This combination of AutoML and MLOps is driving broader adoption across industries.[13, 38]

As explained in[38] AutoML (Automated Machine Learning) refers to the automation of the various steps involved in creating a machine learning model, such as data preparation, model selection,

supporting services. According to them[4], tool diversity is highest in Storage/Versioning and CI/CD categories, while components like Model Repository and ML Model Deployer show the most varied tool implementations. Several components including Dataset Catalogue, Raw Data Store, and various monitoring elements currently lack specific tool mappings in the analyzed literature.

Figure 1.11: MLOps architecture components[4]



A prototype system was developed by [33] to validate their proposed architecture for stream learning (SL) in production-like conditions. Implemented as containerized Python microservices managed by Kubernetes, the system includes three core modules: Inference Endpoints, Model Versioning, and Model Update. Inference Endpoints handle scalable model inference using APIs, and regularly check for updated models. Model Versioning stores SL model versions using MLFlow, serving them to endpoints. Model Update uses Apache Kafka for ingesting labeled events and incrementally updates models, periodically creating new versions. The architecture supports horizontal scaling, seamless model updates without downtime, efficient version management, and is portable for cloud or on-premise deployment. It improves adaptability to evolving data while maintaining performance and stability.

Specialized platform examples

The Acumos AI platform[28] is an open-source project that propose a platform that allows to hotswap models and deploy them to wellknown cloud providers (targets like Kubernetes, AWS, Azure or GCP). By using microservices, they made it easier to swap the models hidden behind a REST API.

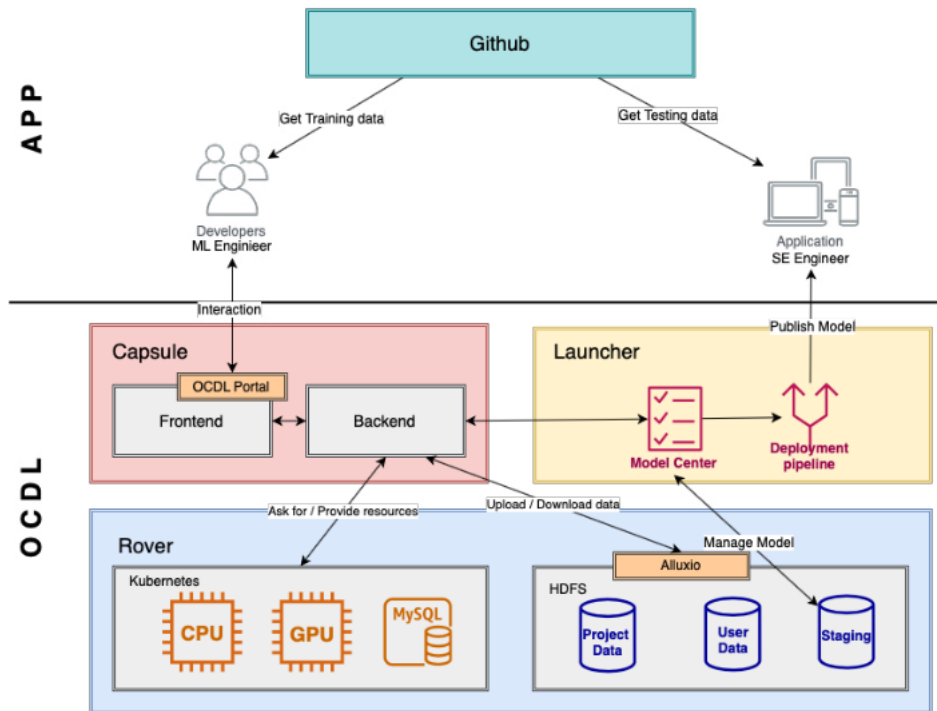
MLOps architectures increasingly adopt microservices patterns for model deployment, enabling independent scaling, versioning, and management of individual models. This approach facilitates A/B testing, canary deployments, and rollback procedures while supporting diverse model types and serving requirements within the same infrastructure[13, 26].

The Nifi project[27] uses the Acumos project to demonstrate how to take an ML model to production. The Nifi project implement automatic and reusable ML model training and ML model serving pipelines.

According to [28], Reusable MLOps is still a novel concept.

The OCDL platform[23] was built on Kubernetes and other open-source frameworks to provide a scalable solution. It also integrates with GitHub to enable CI/CD pipelines and delivery, as illustrated in Figure 1.12. OCDL provides an online IDE for model coding, a reusable code template, data access and GPU/CPU computing instances[23] .

Figure 1.12: The OCDL architecture layers[23]



We will use all their remarks and experiment to create our own in the next chapter.

1.5.6 DataOps Integration with MLOps

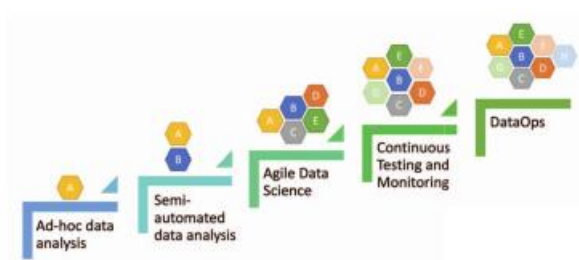
The integration of DataOps with MLOps creates a seamless workflow from raw data to deployed ML models. This integration addresses several critical challenges[31]: DataOps serves as the foundation

for MLOps by ensuring robust and reliable data processes that support the entire machine learning lifecycle. It enables effective data preparation by cleaning, transforming, and validating data before it enters ML pipelines. Through feature engineering, DataOps helps convert raw data into meaningful inputs that improve model performance. Data versioning is also a key component, allowing teams to track data lineage and ensure the reproducibility of machine learning experiments. Additionally, data quality monitoring plays a vital role in detecting data drift or anomalies that could negatively impact model outcomes, ensuring models remain accurate and trustworthy over time.

According to [35], organizations that implement integrated DataOps and MLOps practices realize significant advantages in operational efficiency.

The evolution from ad hoc data analytics towards DataOps shows (fig: 1.13) it follows the path to more automation and is similar to the MLOps evolution we'll see later (fig: 1.14).

Figure 1.13: DataOps Evolution[31]



1.5.7 Maturity Models

The authors of [17], they propose a maturity model that goes from manual MLOps to fully automated MLOps. It differs from the Microsoft and the Google MLOps maturity models, but they all tend to go from manual to fully automated MLOps processes using automated CI/CD and ML pipelines. [38]

In their research [12] also identifies key motivators for MLOps and proposes three levels of MLOps maturity. Level 3 being the most sophisticated and recommended for enterprise solutions.

It expands on Google maturity model [25] which involves 3 levels: Manual process, ML pipeline automation, CI/CD automation. In this recent paper [37], they compare different maturity models for MLOps while introducing LLMOPs (Large Language Model Operations).

1.5.8 GitOps in MLOps workflow

This paper [30] propose an MLOps workflow that follows the GitOps principles. They explain that to maintain the GitOps approach, they have established a separate git repository where they push their configuration changes (e.g., Docker image tags) during a step in their GitHub Action pipeline. They use ArgoCD to sync this repository with predefined environments (development, staging and production).

Figure 1.14: Maturity levels [17]

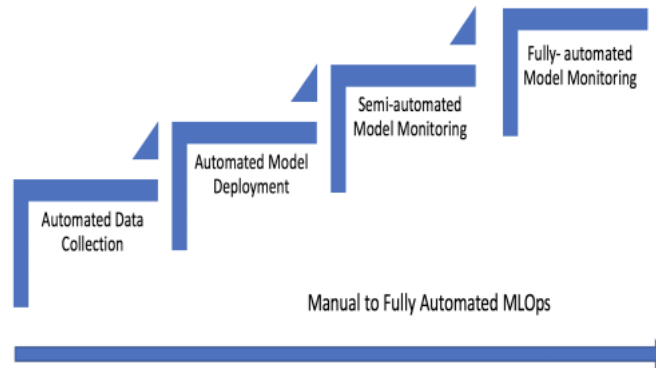
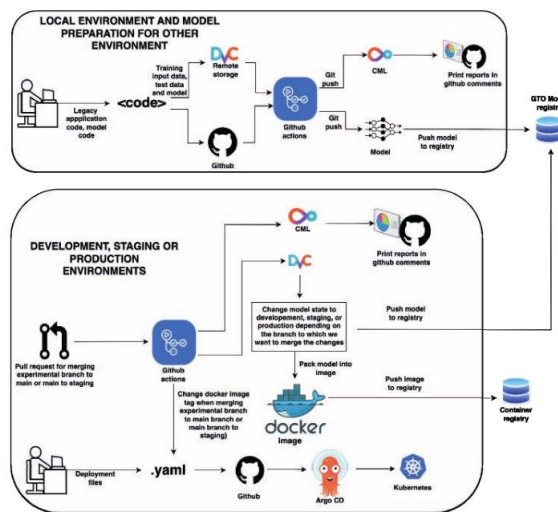


Figure 1.15: Proposed MLOps/GitOps infrastructure[30]



1.6 Tools and Platforms

In this section we will present some of the tools that the literature recommend to implement MLOps workflow.

1.6.1 Version Control and CI/CD Pipelines

Majority of the article we found in the literature are using Git and particularly GitHub[14] as their collaborative development platform. Github uses Github Actions to define their CI/CD pipelines. It offers the possibility to host self-hosted runners to run private pipelines on premise. Azure devops, Gitlab, Gitea and Bamboo pipelines can also be found in some articles[19]

1.6.2 Apache Airflow

Apache Airflow[2] is a platform created by the community to programmatically author, schedule and monitor workflows[2]. It's a general pipeline tool that can be deployed on Kubernetes. It can also be used for ML workflow orchestration[19].

Defining a general MLOps workflow with explicit input-output interfaces is a good practice because it reduces common failure modes and improves clarity. Tools like Apache Airflow can then better manage execution by identifying independent tasks and running them in parallel, leading to more efficient and reliable workflows.[8] It can be used as a DataOps/MLOps workflow orchestrator[16, 8].

An Airflow *DAG* (Directed Acyclic Graph) is the core concept of Airflow[19, 2]. It's used to define pipelines as tasks that are organized with relationships.

1.6.3 Kubeflow

Kubeflow[20] is an open-source platform designed to simplify the deployment, management, and scaling of machine learning workflows on Kubernetes. It provides tools for building portable, scalable, and efficient ML pipelines, including components for training, hyperparameter tuning, and serving models. Kubeflow is used or mention in many publication and MLOps platforms[12, 42, 19]

Kubeflow is a platform with 6 components that can be used as standalone component[20].

- Pipelines (KFP) to implement and run MLOps pipelines.
- Notebooks for web based development.
- Kubeflow Central Dashboard integrate together all Kubeflow components within a single web interface.
- Model Training (Kubeflow Trainer) to ease integration with python machine learning frameworks like tensorflow and more.
- AutoML (Katib)
- Model Serving (Kserve) to help delivery and deployments to production.

Kubeflow offer a Domain Specific Language as a python library to orchestrate the workflow. Kubeflow pipeline integrate with S3-storage (Amazon Simple Storage Service) which is a standard storage solution used as a model repository like in[7]. We'll investigate the kubeflow pipelines and storage later in this paper.

1.6.4 Databricks

Is another platform to orchestrate MLOps workflows. It's a Data-centric approach to establish and scale machine learning specialized into MLOps for large language models LLM[7].

Databricks is a unified data analytics platform that integrates data engineering, data science, and machine learning workflows.

In an MLOps context, It provides an end-to-end MLOps platform with MLflow integration for experiment tracking, model registry, and deployment Features automated CI/CD pipelines specifically optimized for ML workflows Offers Delta Lake for reliable data management with ACID transactions Includes feature store capabilities for feature engineering and sharing Supports both batch and real-time serving for ML models[19, 8].

1.6.5 Argo

[5] It's a suite of open-source tools for Kubernetes, designed to improve workflows, deployments, and continuous delivery.

We are particularly interested with ArgoCD and Argo Rollouts which are both mentioned in the literature[12]. Kubeflow uses Argo workflow for its pipeline orchestration capabilities.

- ArgoCD is a declarative, GitOps-based continuous delivery tool for Kubernetes. It automates application deployments by syncing the desired state defined in a Git repository with the actual state in the cluster, ensuring consistency and enabling easy rollbacks.
- Argo Rollouts is a Kubernetes controller that provides advanced deployment strategies like canary and blue-green rollouts. It allows for progressive delivery, enabling safer and more controlled updates by gradually shifting traffic to new versions while monitoring their performance.
- Argo Workflow is a workflow orchestrator for Kubernetes. It's used by Kubeflow to orchestrate MLOps pipelines.

1.6.6 MLflow

MLflow is an open-source platform for managing the ML lifecycle, including experimentation, reproducibility, deployment, and a central model registry.[8]

MLflow provides an advanced experiment tracking functionality, a model registry, and model serving component[19].

1.6.7 ZenML

ZenML is an extensible, open-source MLOps framework for creating portable, production-ready ML pipelines cited by[29]. ZenML is designed to enable collaboration between Data Scientists, ML Engineers, and MLOps Developers throughout the development-to-production process[6]. It provides interfaces and abstractions specifically tailored for machine learning workflows.

1.6.8 TensorFlow Extended

TensorFlow Extended provides a library to define each tasks of an end-to-end ML pipeline[19].

1.6.9 Dagster

Dagster is a data orchestrator designed to manage data assets like tables, ML models, and reports throughout the entire data development lifecycle—from local development to production[6]. It supports orchestration of various ML pipelines (e.g., feature engineering, training, batch inference) but compared to Prefect, another orchestrator, Dagster emphasizes a full-lifecycle, first-principles approach to data engineering enabling seamless deployment across different infrastructures[6].

1.7 Results

As a result of our literature review, we compiled in the table 1.1 the tools presented earlier with a focus on their primary purpose regarding DevOps, MLOps or DataOps. We wanted to know how much they were cited by the literature and what are their deployment focus either cloud based or Kubernetes focused.

Table 1.1: Tools Support and references

Tool	Primary Purpose	Kubernetes Support	Cloud Support	Primary Focus	Cited by
Kubeflow	End-to-end ML platform on Kubernetes	Native	Multi-cloud	MLOps	[10, 19, 39, 12, 38]
MLflow	ML lifecycle management	Limited	Multi-cloud	MLOps	[33, 8, 15, 19, 7, 12]
Argo	Kubernetes workflow orchestration	Native	Multi-cloud	DevOps	[12]
ZenML	Portable ML pipelines	Good	Multi-cloud	MLOps	[]
Airflow	General workflow orchestration	Good	Multi-cloud	DataOps	[8, 19, 7]
TensorFlow Extended (TFX)	Production ML pipelines	Good	Multi-cloud (GCP focused)	MLOps	[19]
BentoML	Model serving and deployment	Good	Multi-cloud	MLOps	[39, 7, 38]

Tool	Primary Purpose	Kubernetes Support	Cloud Support	Primary Focus	Cited by
Dagster	Data Orchestrator	DataOps	[6]		
GitHub	Version control and CI/CD	Good	Multi-cloud	DevOps	[26, 30]
Azure DevOps	Microsoft DevOps platform	Good	Multi-cloud (Azure focused)	DevOps	[19, 13]
GitLab	DevOps platform with Git	Good	Multi-cloud	DevOps	[39]

Based on our research we also propose this checklist of components following the good practices reviewed before. We will check every part of this list in our implementation in the next chapter.

Good Practice	Tool Used (examples)
CI/CD automation	GitHub Actions
Pipeline/Workflow orchestration	Kubeflow Pipelines, Apache Airflow
Reproducibility using pipeline orchestrator	Kubeflow, Airflow
Versioning of data, code and model with model registry, code repositories and container registries	GitHub, Docker Hub
Collaboration with collaborative development platform (GitHub, Gitlab, Azure Devops)	GitHub
Continuous ML training and evaluation with well-defined triggers for pipelines	Apache Airflow, Kubeflow Pipelines
ML metadata tracking	Kubeflow
Continuous Monitoring (Monitoring system)	OpenSearch
Feedback loops and Data Drift Detectors, Human Verifications	

1.8 Conclusion

During our research and thanks to the survey we conducted we gain enough knowledge and confidence that our project was indeed of some usage.

As shown and stated in our results we can design an MLOps workflow thanks to already existing literature and tools. But we didn't find any complete standard workflow that reunite Devops CI/CD pipelines and MLOps pipelines using traditional Git Branching strategies. Tho we found GitOps cited in the literature about MLOps, we only find one articles with a proposed implementation to define a MLOps workflow following the GitOps principles[30]. We'll follow their remarks, and we'll try in our project to implement a MLOps/GitOps workflow using state-of-the-art frameworks and tools.

MLOps is stated to be the most efficient way to integrate ML models into production, with continuous training and mature systems leading to more realistic and effective models.[12]

The integration of machine learning systems into production environments presents unique challenges that extend well beyond traditional software development practices. As organizations increasingly rely on machine learning models to drive critical business decisions and user experiences, the gap between experimental model development and robust production deployment has become a significant barrier to realizing value from artificial intelligence investments[15]. This gap has given rise to MLOps—a discipline combining machine learning, DevOps, and data engineering practices to streamline the end-to-end machine learning lifecycle[19]. Recent studies have highlighted the difficulties organizations face when transitioning machine learning from research to production.

Chapter 2

Thesis project: Implementing a MLOps/GitOps workflow

2.1 Introduction

Building upon our state-of-the-art review of MLOps practices, this chapter introduces our practical implementation of a MLOps/GitOps workflow. The proposed workflow is designed with versatility in mind—capable of enhancing existing projects at various maturity levels or providing a robust foundation for new initiatives seeking accelerated production deployment.

Rather than developing yet another platform, our approach integrates established tools and workflows identified in the literature into a cohesive system tailored to address modern machine learning development challenges. A distinguishing feature of our implementation is the incorporation of GitOps principles into the MLOps workflow, creating a synergistic relationship between these complementary methodologies.

At the core of our design is the GitOps philosophy, which establishes Git repositories as the single source of truth for code, configurations, and infrastructure. We leverage GitHub repositories as the central foundation, with GitHub Actions orchestrating our DevOps workflows. This integration extends further through tools such as ArgoCD, GitSync, and custom scripts that trigger our DataOps and MLOps pipelines.

Our infrastructure is deployable across one or multiple Kubernetes clusters, depending on specific project requirements. This architecture decouples the workflow definition from hardware considerations, offering significant flexibility. The Kubernetes implementation supports dedicated nodes with specialized resources (high CPU, memory, GPU) for compute-intensive machine learning or data processing operations. Furthermore, this approach accommodates diverse deployment scenarios:

- Cloud-based Kubernetes clusters
- On-premises infrastructure

- Single-node Kubernetes configurations running on development laptops (within hardware constraints)

The seamless integration of DevOps, DataOps, and MLOps pipelines culminates in a comprehensive MLOps workflow that is portable across any Kubernetes environment. This integration addresses the full machine learning lifecycle from development to deployment while maintaining consistent practices.

Our implementation was specifically developed to support our promoter’s ongoing LSFB (Langue des Signes de Belgique Francophone) project, which is already operational in a production environment. The practical application of our workflow to an existing, real-world machine learning system provided valuable insights into the challenges of MLOps adoption beyond theoretical constructs. By enhancing the established LSFB project infrastructure, we could demonstrate the adaptability and incremental benefits of our approach while supporting continued model development and improvement for this important initiative.

In the following sections, we detail our use cases and the specific tools selected to implement our workflow and explain how they interconnect to create a modular system that can adapt as projects evolve in complexity and maturity. Notably, our tool selection was significantly influenced by the existing LSFB project infrastructure. The current production system consists of a Python-based model deployed as a Docker image that exposes an API consumed by a web frontend. The project already utilizes GitHub for code management and Airflow for certain pipeline operations, which represents partial implementation of practices identified in our literature review. These existing elements served as foundational components that our comprehensive workflow needed to accommodate and enhance rather than replace, demonstrating the practical flexibility of our approach.

2.2 Use Case Definitions

Before proceeding further, we formally outline the use cases this project aims to address. We will describe our use case diagram displayed in figure 2.1.

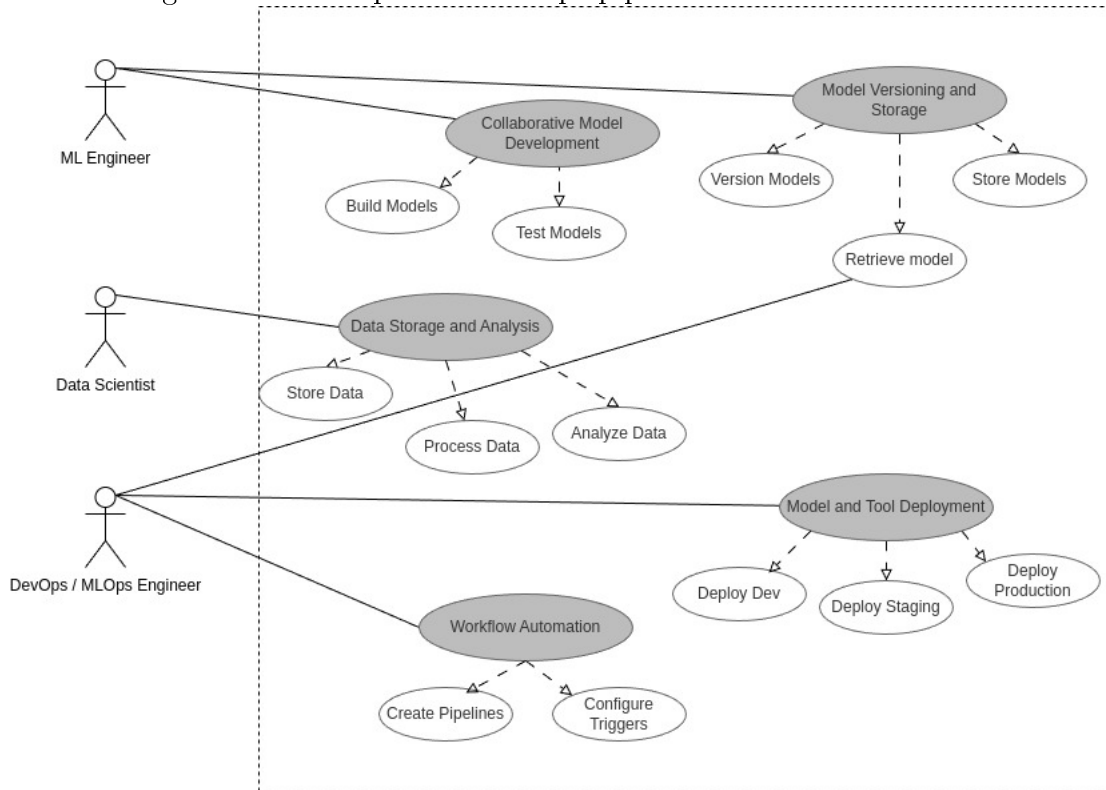
2.2.1 Collaborative Model Development

The platform must provide the necessary tools and infrastructure to enable development teams to collaboratively build, test, and deploy models across segregated environments.

2.2.2 Data Storage and Analysis

A robust system for storing, processing, and analyzing data is required to support model training and evaluation.

Figure 2.1: Example of a MLOps pipeline define within Airflow



2.2.3 Model Versioning and Storage

The solution must include a structured approach to versioning, storing, and retrieving trained models efficiently.

2.2.4 Model and Tool Deployment

The system must support seamless deployment of models and associated tooling across environments—from local development setups to production—with minimal friction.

2.2.5 Workflow Automation

To enhance operational maturity, we prioritize automating repetitive tasks in the workflow. This will be achieved through well-defined pipelines and triggers, informed by our state-of-the-art review.

2.3 Tools Selection Rationale

Having established the theoretical foundations and component descriptions in our state-of-the-art review, this section focuses specifically on our tool selection rationale. While our workflow is designed to be fundamentally tool-agnostic, the following choices were made to address the specific requirements of our implementation context, particularly the existing LSFB project infrastructure.

2.3.1 Python

The foundational programming language for our entire workflow implementation, selected primarily to maintain compatibility with the LSFB project’s existing codebase and the broader machine learning ecosystem. The project’s current use of Keras for model development and our demonstration implementations using scikit-learn both benefit from Python’s extensive machine learning library ecosystem and developer familiarity. Notably, Python’s pervasive role extends to our pipeline implementations, as both Airflow DAGs and Kubeflow pipelines are defined using Python, creating a consistent development environment across all workflow components.

2.3.2 Git Repositories and CI/CD

We selected GitHub as our version control and CI/CD platform primarily due to its established presence in the LSFB project ecosystem. This choice provides continuity with existing development practices while enabling us to leverage GitHub Actions for workflow automation without introducing additional integration complexity. GitHub’s robust API and extensive integration capabilities further support our goal of building an interconnected MLOps workflow.

2.3.3 IAC, Containers and Registry

Docker was the natural containerization choice given the LSFB project's existing Docker-based model deployment. DockerHub serves as our container registry and Helm chart repository, offering reliable accessibility and established integration paths with our other selected tools. This approach maintains compatibility with the current production environment while enabling more sophisticated deployment patterns.

2.3.4 Kubernetes

Kubernetes was selected as our orchestration platform for its exceptional flexibility and robust ecosystem. Its ability to operate across various infrastructure environments (cloud, on-premises, development workstations) directly supports our portability requirements. Furthermore, Kubernetes provides the foundation for advanced deployment strategies needed in ML systems and enables seamless integration with specialized tools like KubeFlow and Airflow.

2.3.5 Airflow

We chose Airflow for DataOps orchestration to maintain compatibility with existing pipelines in the LSFB project. This selection allows us to extend current functionality while providing a bridge to new MLOps capabilities. Airflow's Python-based DAG definitions integrate naturally with our development workflow and allow data scientists to define complex pipelines using familiar syntax. Airflow's ability to trigger KubeFlow pipelines creates a unified workflow that respects team boundaries and specialized tooling preferences while maintaining end-to-end process integrity. Moreover, by leveraging Airflow's recent GitSync capabilities, we were able to successfully implement GitOps principles within our MLOps project.

2.3.6 KubeFlow

KubeFlow was selected primarily for its storage solutions and pipeline capabilities that directly address ML-specific workflow requirements. The Python SDK for KubeFlow Pipelines enables seamless integration with our existing Python codebase, allowing model developers to define reproducible ML workflows using familiar programming patterns. This choice provides a growth path for the LSFB project, allowing incremental adoption of additional MLOps features as project maturity increases, without requiring architectural redesign.

2.3.7 ArgoCD

ArgoCD was chosen as our GitOps implementation tool for its native Kubernetes integration and declarative approach to deployment automation. This selection enables us to maintain

infrastructure and application configurations as code within our GitHub repositories. Additionally, Argo Rollouts provides sophisticated deployment capabilities essential for our model updates in any environments.

ArgoCD can be configured to trigger integration tests defined via helm test. Its notification service may be used to send webhooks to GitHub upon successful deployment. This contributes to a more fully automated workflow.

Additionally, ArgoCD’s project abstraction enables fine-grained, role-based access control (RBAC), allowing developers to deploy only predefined resources within authorized clusters and namespaces. When combined with Kubernetes resource quotas, this approach empowers the infrastructure team to enforce secure and isolated development environments.

2.4 Infrastructure and Workflow

Having established the tool requirements, we now detail the implementation of our workflow along with the minimal infrastructure necessary to sustain it.

In figure 2.2, we describe our infrastructure that can be first deployed on a local Kubernetes single node cluster with independent DataOps and MLOps pipelines develop by potentially different teams with different roles.

By using Airflow pipelines as general pipelines to trigger our Kubeflow pipelines, we can easily gain in maturity towards more automation by combining the DataOps pipelines and MLOps pipelines together when they are mature enough.

This way we enable kubeflow features for our model developers within a more general purpose environment.

By using the git-sync capabilities of Airflow we can synchronise our dags directly with airflow and even trigger them automatically in a later stage towards automation.

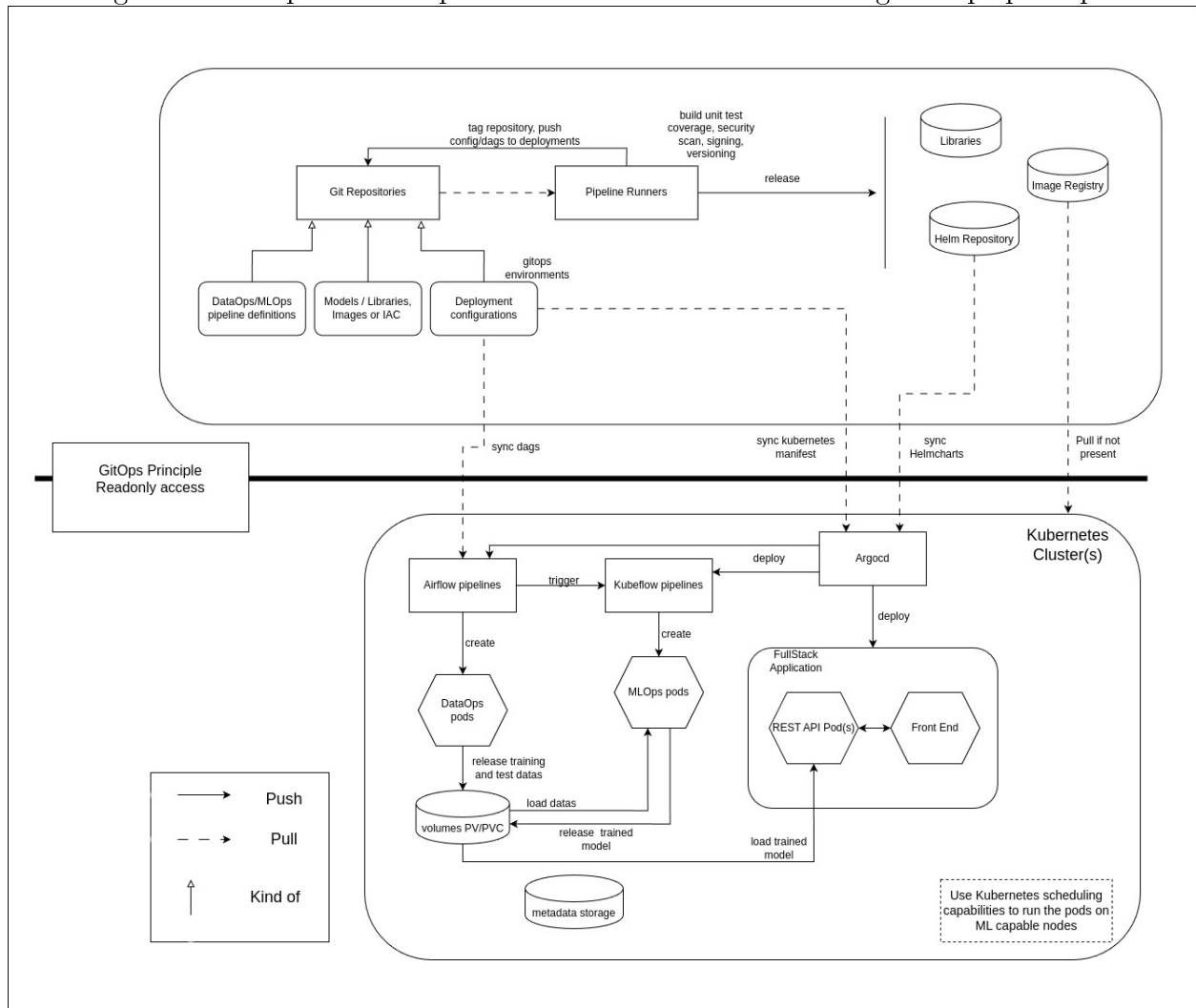
Notably our infrastructure shows the GitOps pull based strategy. Only the GitHub actions runners do pushes to other git repositories or other storage like HelmChart repository, images repository or any library repository. Our Kubernetes environments only pull changes from those repositories, making our infrastructure fully private and easily deployable anywhere. We will now further describe the component of our workflow.

2.4.1 General workflow

As previously outlined in our state-of-the-art review, the primary triggers for initiating our workflow are either the emergence of a new development need or the availability of new data. Beyond the initial phase—during which the project is discussed and defined in collaboration with business stakeholders. The motivation for further development or model retraining typically arises from the system’s monitoring and feedback loops.

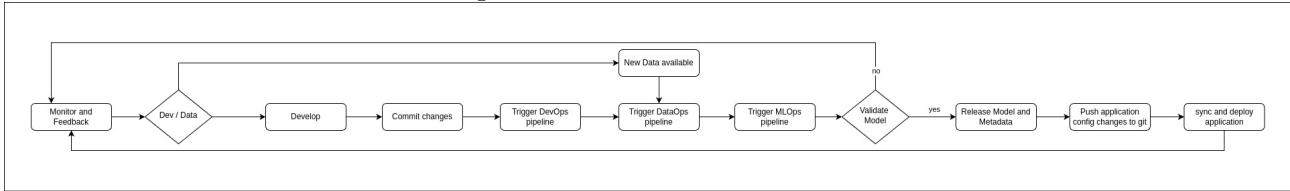
Within our GitOps-based MLOps workflow, any change whether in code or configuration must be committed to the appropriate Git repository. Each push to the repository triggers

Figure 2.2: Proposed MLOps Kubernetes infrastructure using GitOps principles



a DevOps pipeline, responsible for building images, running unit tests and releasing images. This part culminates in updating configuration files in a dedicated configuration repository. This repository is continuously monitored and synchronised by ArgoCD or Airflow, which applies the changes to the cluster and initiates the DataOps pipeline or integration tests. Upon successful completion of this stage, the MLOps pipeline is triggered. If the newly trained model passes validation criteria, it is released to a model repository. In accordance with GitOps principles, the deployment involves updating configuration files in a separate Git repository, enabling ArgoCD to automatically deploy the new model version to the production environment thus closing the automation loop as shown in figure 2.3. By applying a GitOps strategy to both our development projects and infrastructure as code, we establish a fully GitOps-driven infrastructure that can consistently deploy our applications and infrastructure across any environment.

Figure 2.3: General Workflow



2.4.2 Git Repositories

As mentioned, we use Git repositories for version control and to trigger our DevOps pipelines. Additionally, they serve as a way to distribute responsibilities and coordinate work across teams within our MLOps project. Using GitHub's access management features, we are able to effectively manage these workflows and permissions.

Within our infrastructure, we consider 4 types of Git Repositories: Code, MLOps/DataOps pipelines, Deployment/Configuration, DevOps pipelines templates.

Following GitOps principle those are the source of truth for all our code and configurations, for the infrastructure, the pipelines, the model and the software development.

Code Repositories

that holds code for models, libraries, docker images and infrastructure as code using Helm.

MLOps and DataOps Pipelines Repositories

In our infrastructure those repositories holds the definition of our Airflow DAGs. In the first iteration those repositories can also hold images for.

We adopt a consistent structure (figure 2.4) for both our DataOps and MLOps pipeline repositories, incorporating a `config.yml` file that mirrors the organizational pattern of Helm chart project (`values.yml`). In both cases, an `images` directory is used to define containerized steps for the pipelines and for deployment configurations within the Helm charts. This consistency permit to reuse our Devops pipelines templates.

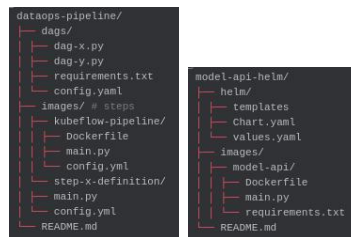


Figure 2.4: Consistent structure within repositories

Deployment/Configuration repositories

Those repositories are used to hold configuration for the deployed applications and Airflow dags. We use ArgoCD GitOps implementation to synchronise changes to those repositories. Airflow git/sync feature allows us to synchronise our dags with Change in those repositories can be automated by the pipeline runners. Depending on your team and organization those repositories can be separated into multiple repositories (per team, per domain, per environment (dev,test,staging,prod)) For the Dev environment we allow developers to push from their code repositories within those repositories. For the production environment we use a pipeline to pull the configuration from the staging environment. Production environments can be manage by an Operation team to approve and manage those deployment according to organization policies. We called this promote our configuration to a new environment. In case of full automation the pulling/promotion can be trigger automatically by ArgoCD sending webhooks on any test results desired.

CI/CD pipelines templates

In those we define DevOps workflows that can be used in any repository to be used by the developers. It includes building image workflows, versioning with tags on repositories, pushing new configurations into deployments repositories. By defining them in a separate repository it allows us to version them and make it easy for developers to choose a workflow. Each workflow is closely tight to the structure of the repository, so we defined one per type of repositories.

General Rules for Managing our Git Repositories

We follow the GitHub flow with adaptations to fit our workflow. However, these rules can be adjusted as needed, and our project is not strictly bound to them. They are necessary, though, to enable automation alongside human review through approval gates.

- A Pull Request is required for merging into the `main` branch, including a pipeline run and team review.
- Follow GitHub flow, using `feature/` and `hotfix/` branches for development.
- Approvals are required for deploying to environment-specific repositories.
- The `main` branch is deployed to the production environment or released into production Docker/Helm repositories.
- Other branches are deployed sequentially to all environments, passing tests and approval gates before merging to main.

2.4.3 GitOps

With a Helm-based installation pipeline, the runner requires write access to push applications directly to Kubernetes. By adopting ArgoCD and a GitOps approach, we eliminate this requirement by pulling changes directly from GitHub. This enables all operations to be managed within GitHub. However, it necessitates properly configured permissions in GitHub to prevent potential security breaches.

In our project, for consistency, we use Airflow's integrated Git-Sync feature along with custom scripts to load and trigger DAGs within Airflow, similar to how ArgoCD manages deployments.

Figure 2.5 shows a screenshot of our deployed infrastructure as displayed in the ArgoCD web interface.

We used the official Airflow Helm chart, along with a companion application to define additional dependencies such as ingress, service accounts, and volumes.

We also deployed Kubeflow Pipelines using Kustomize, along with its required dependencies, in a separate ArgoCD application.

Our custom REST API, `mthmlops-app`, which hosts the machine learning model, was deployed as well.

Implementation details and usage instructions can be found in the corresponding GitHub repositories (refer to the annexes for links). To support this setup, we created a dedicated ArgoCD project, allowing us to separate this ML workflow from other ML development projects. Figure 2.6 displays our parent application, illustrating the use of the ArgoCD app-of-apps pattern.

Figure 2.5: Infrastructure deployed by ArgoCD

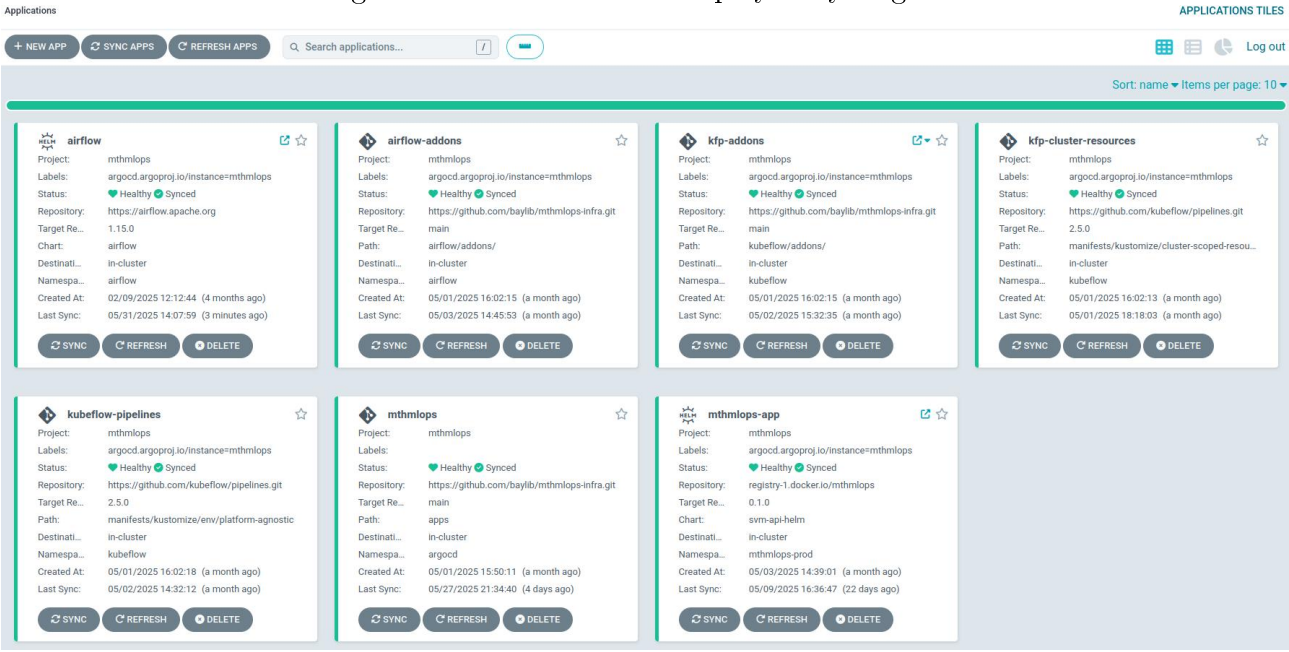
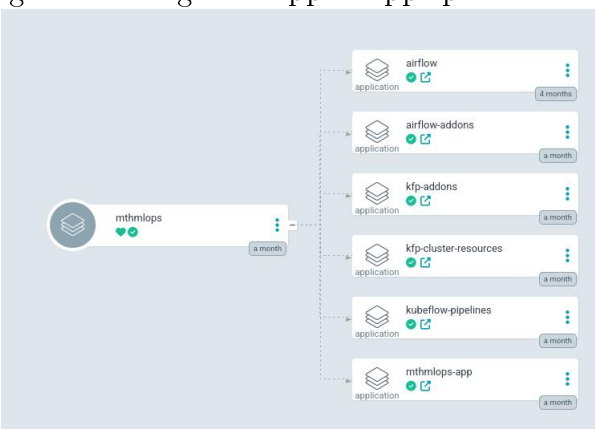


Figure 2.6: ArgoCD App-of-apps pattern usage



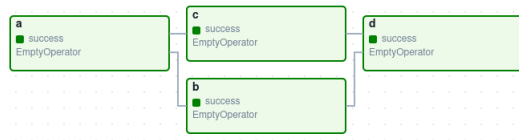
2.4.4 DataOps pipelines with Airflow

DataOps pipelines are defined as Airflow DAGs that leverage Airflow’s `KubernetesExecutor`, which creates a pod (i.e., a running container) for each step defined in the pipeline.

The goal is to deliver usable data into accessible storage for our Machine Learning Engineering teams. In the initial iteration or development environment, this can be a local volume, with migration to external storage volumes planned later. By using Kubernetes Persistent Volumes (PV) and Persistent Volume Claims (PVC), we can mount external S3-compatible storage in the same way. Airflow’s Domain-Specific-Language also facilitates easier data storage management.

Steps in our DataOps pipeline can be integrated with an observability stack such as OpenSearch or Elasticsearch to analyze and store data. These tools can also serve as monitoring solutions for our entire infrastructure.

Figure 2.7: Example of a pipeline define within Airflow



2.4.5 MLOps pipelines with Kubeflow

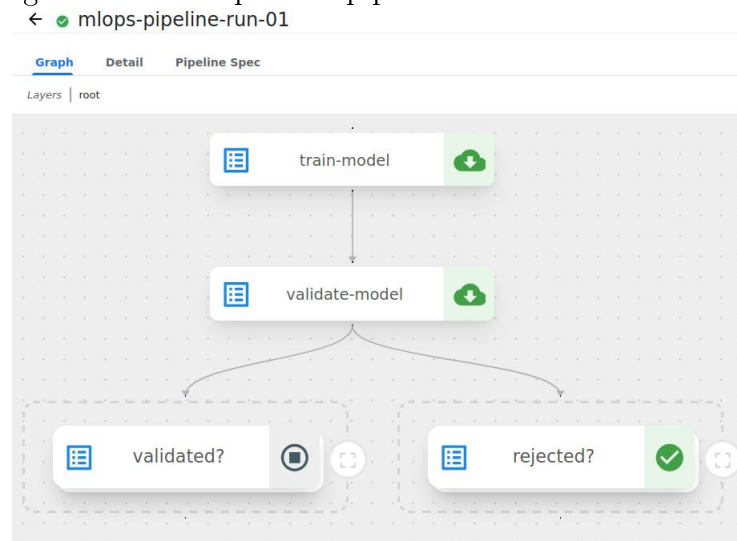
Within our infrastructure (see Figure 2.2), Kubeflow enables us to deploy all the necessary steps of our MLOps pipeline by creating pods in the appropriate namespaces within our Kubernetes cluster—much like how Airflow manages our DataOps pipelines. We use Kubeflow to provide our ML engineers with additional features such as model and metadata storage. Furthermore, other components from the Kubeflow ecosystem can be integrated at any time if needed.

2.4.6 Storages

Here we describe the storage components in our infrastructure (Figure 2.2) and their roles within our workflow.

- **Model repository:** Managed by Kubeflow, this S3-compatible storage holds trained, pretrained, and untrained models. It can be mounted as a volume in Kubernetes and attached to running containers.
- **Image repository:** Stores container images that our cluster pulls during deployment.
- **Helm Chart repository:** Maintains and versions our infrastructure-as-code templates as Helm packages.

Figure 2.8: Example of a pipeline define within KubeFlow



- **Metadata database:** Managed by KubeFlow, it stores model metadata and experiment tracking information.
- **Git repositories:** Store code and configuration, as described earlier.
- **Local volumes:** Used for storing other implementation-specific artifacts.

2.4.7 Model development

As previously mentioned, we use Helm and Docker to package our model code. In this section, we'll go into detail about how the model is defined. The model's Docker image supports three operational modes, each configurable via parameters:

- Training mode that loads and trains the untrained model on training data.
- A validation mode to load, test and validate the model.
- A listen mode define as a REST API server to interface with external frontends or services.

To integrate smoothly with our DataOps and MLOps pipelines, we define parameters that specify storage locations. This ensures that any container image meeting these requirements can be seamlessly used within the pipeline. Since we use the container layered model, it allows us to take any base image and add a layer that satisfies our pipeline's interface requirements. While it's convenient to define all stages (training, validation, and listening) within a single image, it may be preferable to split them into two or three separate images. One downside of

the single-image approach is that the model's code is packaged into the same container that's deployed to production.

To deploy our production REST API server to Kubernetes, we created a standard Helm project with template manifests for a Service, ServiceAccount, Ingress, and Deployment. In the Helm (values.yaml) configuration file, we specify which trained model version should be loaded, allowing for seamless model hot-swapping when needed. For more advanced deployment strategies, we can replace the standard Deployment manifest with a Rollout manifest (Custom Resource Definition (CRD) managed by Argo Rollouts).

To use the model within our MLOps Kubeflow pipeline, we use our training and validation mode and our data location parameters with Kubeflow Domain specific language container components. We use the same approach for our DataOps pipelines steps to ensure consistency as we'll demonstrate later in this paper.

Figure 2.9 shows the model HelmChart and it's component deployed using ArgoCD.

Figure 2.9: Our model REST API HelmChart deployed using ArgoCD



2.5 Roles within our MLOps workflow

Each team can be assigned and linked to our type of Git repositories define earlier^{2.4.2}. All team can receive access to a Deployment/config repository that will be sync to the development environment. DevOps and Operations teams should be responsible for the production repositories.

As we said earlier deployment repositories are synced to its defined environment but can be isolated using kubernetes namespaces, and projects within Airflow, Kubeflow¹ or ArgoCD.

¹require Kubeflow installed in multi-user mode

This will ensure any team will deploy to their own namespaces with they predefined resource quotas.

2.5.1 DevOps Engineers

DevOps engineers are responsible for managing CI/CD pipeline templates, repositories, and the underlying runner infrastructure. They should provide standardized templates for building and releasing Docker images, code libraries, and Helm charts. Additionally, they should establish a process to synchronize configuration changes from a code repository to a deployment/configuration repository.

2.5.2 Data Engineers

The Data Team is responsible for maintaining at least one DAG repository to define their DataOps pipelines. For more complex tasks, they should create dedicated code repositories to build custom images used within their DAGs.

2.5.3 MLOps Engineers

MLOps engineers are tasked with defining the overarching workflow and architecture, as demonstrated in this project. They should oversee and manage relevant tools such as Airflow and Kubeflow, ensuring seamless integration and operation.

2.5.4 Model Engineers

Similar to the Data Team, Model Engineers should maintain at least one DAG repository to define their Kubeflow pipelines and build basic container images. For more advanced steps, such as model and library development or model training, they should create separate repositories. These repositories can follow a release workflow and be integrated as dependencies within the pipeline definitions.

2.5.5 Software Engineers

Software Engineers are responsible for developing the application that integrates and utilizes the model, ensuring it meets functional requirements. This application is defined as a Helm chart within our workflow, enabling deployment through ArgoCD. For advanced deployment strategies, ArgoCD Rollouts can be utilized to ensure more refined and controlled release processes. The trained model version is integrated into a REST API component, which is part of the application.

2.5.6 Example of minimal list of git repositories per roles/teams within an Organisation

Below, we present an example of the tree structure of an organization in GitHub. It is divided into projects containing repositories. Pipeline access between them is managed in GitHub using deployment tokens or SSH keys.

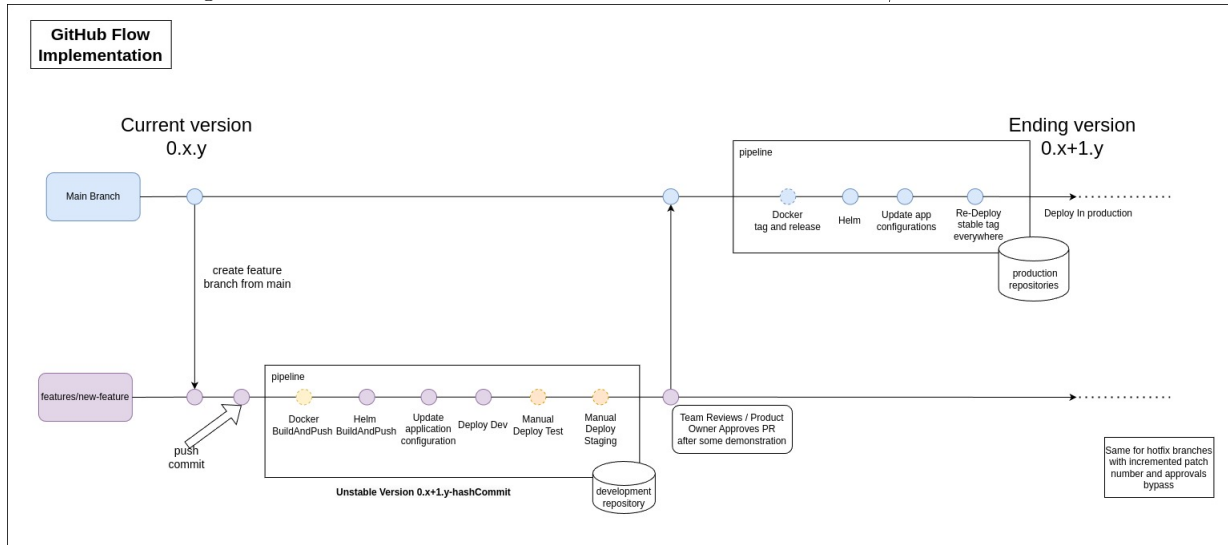
2.6 Workflow pipelines

In this section we will further describe our pipelines definitions.

2.6.1 DevOps CI/CD pipelines

As we explained while describing the infrastructure, part of our global workflow can be implemented within pipelines runners definition. We used GitHub action and the GitHub flow to harmonise the workflow within our different type of git repositories. GitHub allows us to define template in a single repository that can be then versioned and used in other repositories while hiding the complexity of the defined pipelines. We also followed the GitHub flow which involve one main branch and features or hotfix branch to favor small and fast releases. As stated before other strategies can be used with an adapted development workflow.

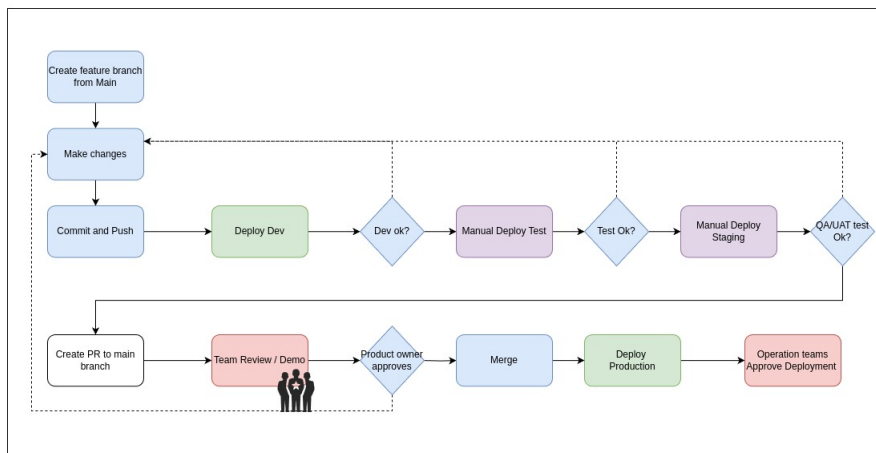
Figure 2.10: Implementation of the GitHub Flow CI/CD workflow



Our DevOps workflow, following the GitHub flow, requires that new development be done on a separate branch. The project version is incremented, and the commit hash is appended to the version string. Unit tests are performed during the build phase within the Docker image

to ensure code quality. This versioned build can then be deployed to each environment, where further tests are executed using ArgoCD and Helm test definitions. Once a merge request is accepted, the commit hash is removed from the version, and the images are either rebuilt or retagged with the new version. Additionally, the repository is tagged to facilitate rollback if necessary. Creating a pull request allows developers to facilitate collaboration and verification before deploying to production. (See Figures 2.10 and ??)

Figure 2.11: Development Team Contribution activity



There are our development pipelines for any git repositories except for the deployment repository that are only targets. Artifact that are not deployed but released to a repository follows the same path but are deployed as dependencies within other projects. Airflow DAGS and Kubeflow pipelines follows the same development workflow and are release into our previously defined git deployment repositories.

We can see in figure 2.12 the implementation of our DevOps pipeline within GitHub. The promotion step involves pushing new configuration to our git repositories that will be synced by ArgoCD or Airflow Git-Sync. We use a Matrix strategy to create build and push jobs for our docker images only if changes were detected in the directory defining this steps. This automation requires squashing commit when we merge to the main branch.

2.6.2 DataOps pipelines

Inspired by our research in the literature we create a sample DataOps pipeline that can be implemented with specific operations.

Figure 2.12: GitHub pipeline implementation running on a feature branch

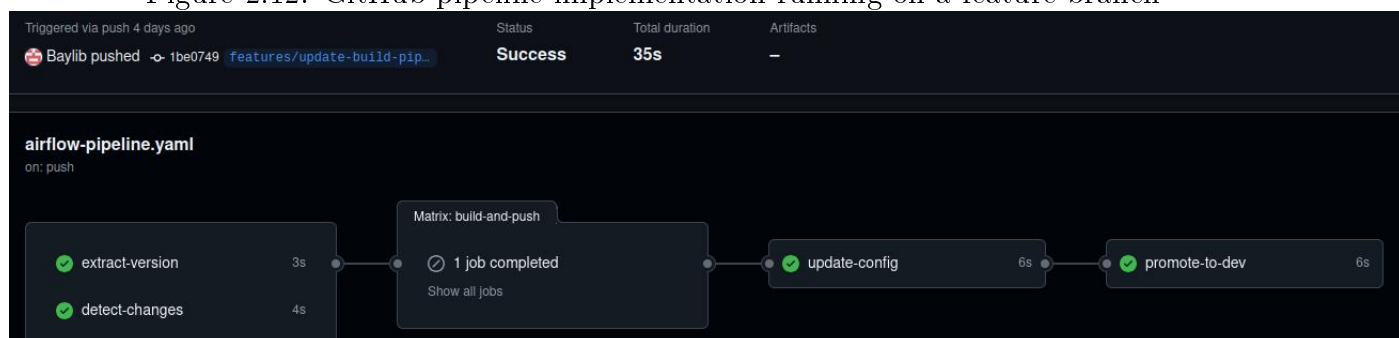
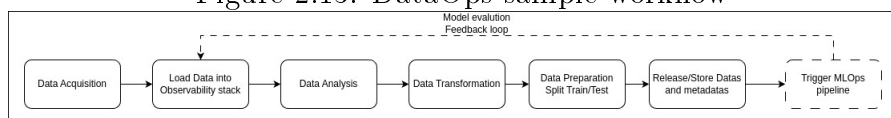


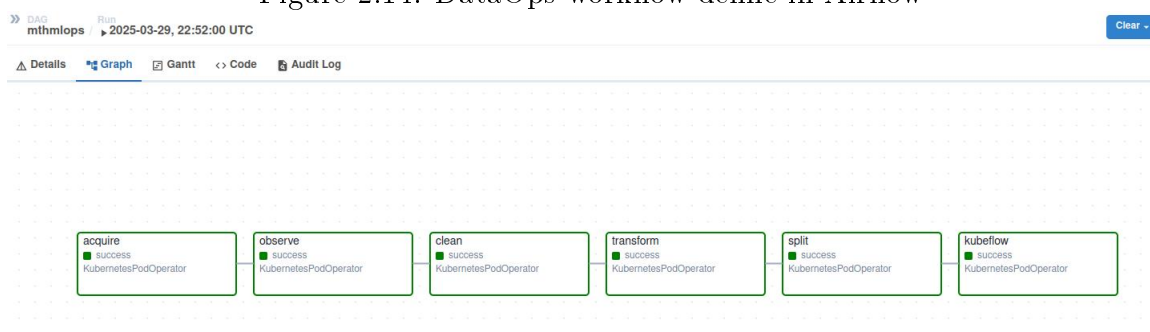
Figure 2.13: DataOps sample workflow



It's defined in our project as an Airflow DAG and can be triggered manually in early stage of the project or automatically when the project has enough maturity.

Each step of the pipeline can be defined as a container image using the Airflow Domain specific language (e.g. ContainerSpec). We use the same parameter definition strategy outlined in our model development section (2.4.7), so that each step in the pipeline can be easily replaced with an alternative implementation.

Figure 2.14: DataOps workflow define in Airflow



Also, by defining all our steps as independent containers, we decouple them from the pipeline orchestrator. This flexibility allows us to implement steps in different languages if better suited for our Data or Machine Learning teams.

One disadvantage is the additional time required to start a new container for each step. However, this overhead is often negligible, as Data and Machine Learning steps typically take

a long time to run.

We demonstrate this flexibility by replacing our original data-collector dummy step with the LSFB dataset library[11], encapsulated within a container. Since the LSFB dataset comes pre-split and prepared, we can skip several steps in our DataOps pipeline.

Below, we provide a detailed explanation of each step in our pipeline.

Acquire

Data is collected from external sources or internal repositories. We use containers with parameters to set appropriate location per project.

Observe

Initial analysis is performed to understand the data's structure and content. In this step we load the data into an observability stack (ElasticSearch or OpenSearch for example)

Clean

Missing values are handled and inconsistencies are corrected.

Transform

Within this step we format the data according to project requirements.

Split

We then divide and release the dataset into training, validation, and test sets.

MLOps pipeline Triggers

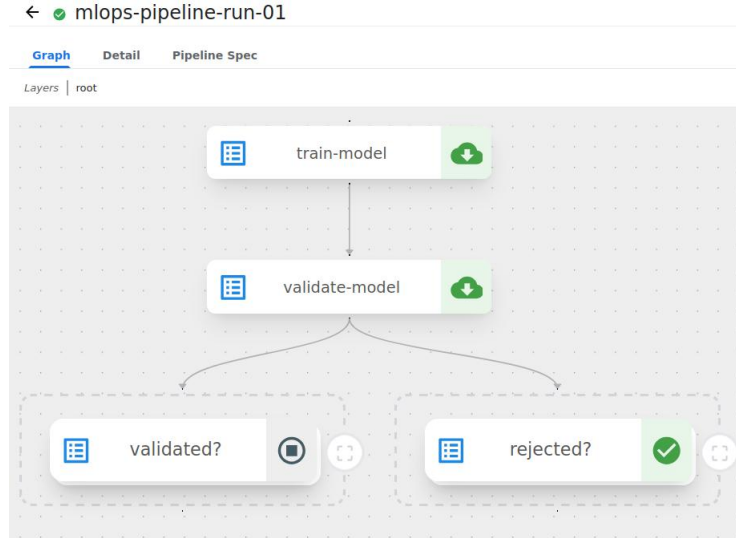
Airflow triggers the Kubeflow pipeline to initiate the MLOps workflow.

2.6.3 MLOps pipelines

Our MLOps pipelines are developed as Kubeflow pipelines with the possibility to trigger them from an airflow DAG even in the early stage of the project. This ensures the possibility of combining it with the DataOps pipeline, facilitating further automation.

In our MLOps workflow, we have established a seamless integration between Apache Airflow and Kubeflow Pipelines to efficiently manage the machine learning lifecycle. This integration automates the process from data availability to model deployment, ensuring a streamlined and reproducible workflow.

Figure 2.15: MLOps workflow in Kubeflow pipelines



Each step of the pipeline is defined as a container image using the Kubeflow Pipelines domain-specific language (e.g., ContainerSpec). We follow the same design principles as in our DataOps pipeline, enabling easy replacement of individual steps and flexibility to use different languages when appropriate. This container-based modularity also ensures less coupling with the pipeline orchestrator.

Model Training

The pipeline begins with a training component that uses the datasets provided by our DataOps pipeline to train a machine learning model.

Model Validation

After training, a validation component assesses the model's performance using the test data. If the model meets our predefined accuracy thresholds and/or performs better than previous models, the workflow proceeds; otherwise, it terminates, ensuring only validated models advance to production.

Release model

Upon successful validation, the pipeline enters the release phase, where the model and its metadata are saved to MinIO. The deployment component updates our production environment with the new model, ensuring minimal disruption and continuous service delivery. The model is released along with its metadata, which currently includes primarily accuracy and versioning information.

Reject the model

If validation fails, the pipeline is halted, and no updates are made to the model or its meta-data. The production environment remains intact, preventing any disruptions and maintaining continuous service.

Although we chose to implement our pipeline this way, a rejection doesn't have to halt the process. Instead, it can trigger model or parameter tuning and start a new training cycle for a defined number of iterations.

Deploy

This part is crucial in our implementation as instead of using Kubeflow serve capabilities we rather use a GitOps methodology and use this step to commit configuration changes in our configuration git repository that is synced by ArgoCD using the deployment strategy define in the deployment or rollout manifest within our HelmChart.

In addition, with our helm tests we can run our integration tests before, during or after the deployment to ensure a successful deployment in any environments.

This way we ensure consistency between all our deployments. ArgoCD notifications can close the loop by confirming the deployment through a webhook call.

2.7 Limitations and Critical Assessment

While our MLOps implementation demonstrates practical value, several aspects allows critical examination.

Architectural Complexity

Our dual use of Kubeflow and Airflow may introduce unnecessary complexity and could potentially be simplified. However, we deliberately maintained this separation for clear role distinction: Airflow handles data orchestration while Kubeflow manages ML-specific workflows. We also found Airflow easier to manage multi-user and namespaces within kubernetes. Future iterations could explore more integrated solutions, particularly as Airflow expands its MLOps capabilities.

Limited Production Validation

Our implementation's real-world validation remains limited. While we demonstrated practical utility through the LSFB dataset pipeline, we only validated the complete ML lifecycle using demonstration models rather than production-grade systems. This restricts generalizability and leaves questions about performance.

Evolving Technology Landscape

The rapidly evolving MLOps ecosystem, including developments like GitHub’s beta model repository, may consolidate functionality that our implementation currently addresses through separate tools. While our containerized approach provides adaptation flexibility, this field requires frequent architectural reassessment.

Despite these limitations, we maintain confidence in our implementation’s core design principles and reusability through state-of-the-art practices including containerization, automated triggers, and well-defined parameters.

2.8 Future Work

This project is currently a prototype suitable for development and testing environments. Future iterations should aim to harden the implementation for production use. While the focus has primarily been on the MLOps workflow, several areas remain to be improved:

- **Production-Ready Infrastructure:** The current setup lacks some of the considerations and robustness needed for a mature DevOps environment.
- **Secret Management:** Integrating a dedicated secrets manager such as HashiCorp Vault would enhance security and secret handling.
- **Improved ArgoCD Usage:** While we adopted the app-of-apps pattern in ArgoCD, enabling ApplicationSets could provide better self-service capabilities for development teams and simplify multi-environment management.
- **Pipeline Orchestration Simplification:** Currently, we use three different orchestrators. However, as platforms evolve—GitHub introducing a model registry, KubeFlow expanding its data management capabilities, and Airflow moving toward deeper MLOps integration—it may become feasible to consolidate to a single orchestrator in the future.
- **Repository Naming:** Standardizing and renaming Git repositories to clearer, more meaningful names will improve maintainability and team onboarding.

These improvements will help transition our prototype into a scalable, production-grade MLOps-GitOps solution.

2.9 Conclusion

During this project, we were able to implement the techniques and concepts learned throughout our master’s program, specializing in areas such as Kubernetes clusters, DevOps, DataOps,

MLOps, Docker, Helm, and Python programming. Through literature research and reviewing various documentation, we successfully implemented an MLOps workflow prototype capable of supporting enterprise-level infrastructure and being reused across multiple data engineering and machine learning projects. We also explored integrating a GitOps methodology with our MLOps workflow, following feedback from our peers.

We answered our research question *"How can a state-of-the-art MLOps-GitOps workflow be implemented or proposed to initiate a machine learning project and support its progression toward greater maturity?"*, by implementing a state-of-the-art MLOps-GitOps workflow that can be used as a project bootstrap and gain maturity over time.

As the project matures, we integrated Airflow and KubeFlow pipelines to enable seamless automation of both our DataOps and MLOps workflows. By applying the same GitOps methodology for managing our DataOps and MLOps pipelines as well as for deploying infrastructure and applications via HelmCharts, we have successfully established a consistent and unified workflow for all development processes, whether for application deployment, Airflow DAG orchestration or KubeFlow pipelines orchestration.

The LSFB dataset provided by our promoters was instrumental in demonstrating the applicability of our workflow, using a real model already in production. Their expertise offered valuable insights and served as a strong foundation for initiating our project.

We integrated all necessary elements for our implementation in accordance with the findings from our state-of-the-art review. To conclude, we have summarized our results in the table below.

Good Practice	Tool Used	Remarks
CI/CD automation	GitHub Actions	We implemented the GitOps principles into our DevOps pipelines
Pipeline/Workflow orchestration	KubeFlow/Airflow Pipelines	We chose a container based approach to ease reusability
Reproducibility using pipeline orchestrator	KubeFlow, Airflow	Both Airflow and KubeFlow DSL ease the experiment tracking and reproducibility
Versioning of data, code and model with model registry, code repositories and container registries	GitHub, Docker Hub	Makes it easy to create free account for student usage
Collaboration with collaborative development platform	GitHub	We specified rules and branch strategy to manage our repositories
Continuous ML training and evaluation with well-defined triggers for pipelines	Apache Airflow, KubeFlow Pipelines	We implemented and demonstrated use of configurable pipelines with clearly defined input/output parameters
ML metadata tracking	KubeFlow	We used KubeFlow DSL to ease metadata management
Continuous Monitoring (Monitoring system)	ElasticSearch	Only tested in early stages of the project
Feedback loops and Data Drift Detectors, Human Verifications	ElasticSearch	Only tested in early stages of the project

Bibliography

- [1] *Airbyte*. URL: <https://airbyte.com/>.
- [2] *Airflow*. URL: <https://airflow.apache.org/>.
- [3] Saleema Amershi et al. “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](https://doi.org/10.1109/ICSE-SEIP.2019.00042).
- [4] Faezeh Amou Najafabadi et al. “An Analysis of MLOps Architectures: A Systematic Mapping Study”. In: *Software Architecture*. Springer Nature Switzerland, 2024, pp. 69–85. ISBN: 9783031707971. DOI: [10.1007/978-3-031-70797-1_5](https://doi.org/10.1007/978-3-031-70797-1_5). URL: http://dx.doi.org/10.1007/978-3-031-70797-1_5.
- [5] *Argo*. URL: <https://argoproj.github.io/>.
- [6] Lisana Berberi et al. “Machine learning operations landscape: platforms and tools”. In: *Artificial Intelligence Review* 58 (Mar. 2025). DOI: [10.1007/s10462-025-11164-3](https://doi.org/10.1007/s10462-025-11164-3).
- [7] Antonio M. Burgueño-Romero, Cristóbal Barba-González, and José F. Aldana-Montes. “Big Data-driven MLOps workflow for annual high-resolution land cover classification models”. In: *Future Generation Computer Systems* 163 (2025), p. 107499. ISSN: 0167-739X. DOI: <https://doi.org/10.1016/j.future.2024.107499>. URL: <https://www.sciencedirect.com/science/article/pii/S0167739X24004631>.
- [8] Andrew Chen et al. “Developments in MLflow: A System to Accelerate the Machine Learning Lifecycle”. In: June 2020, pp. 1–4. DOI: [10.1145/3399579.3399867](https://doi.org/10.1145/3399579.3399867).
- [9] *Docker*. URL: <https://www.docker.com/>.
- [10] Aymen Fannouch, Jihane Gharib, and Youssef Gahi. “Enhancing DataOps practices through innovative collaborative models: A systematic review”. In: *International Journal of Information Management Data Insights* 5.1 (2025), p. 100321. ISSN: 2667-0968. DOI: <https://doi.org/10.1016/j.jjimei.2025.100321>. URL: <https://www.sciencedirect.com/science/article/pii/S2667096825000035>.

- [11] Jérôme Fink et al. “LSFB-CONT and LSFB-ISOL: Two New Datasets for Vision-Based Sign Language Recognition”. In: *2021 International Joint Conference on Neural Networks (IJCNN)*. 2021, pp. 1–8. DOI: [10.1109/IJCNN52387.2021.9534336](https://doi.org/10.1109/IJCNN52387.2021.9534336).
- [12] Satvik Garg et al. “On Continuous Integration / Continuous Delivery for Automated Deployment of Machine Learning Models using MLOps”. In: Dec. 2021, pp. 25–28. DOI: [10.1109/AIKE52691.2021.00010](https://doi.org/10.1109/AIKE52691.2021.00010).
- [13] N. Gift and A. Deza. *Practical MLOps*. O’Reilly Media, 2021. ISBN: 9781098102982. URL: <https://books.google.be/books?id=J99CEAAQBAJ>.
- [14] *GitHub*. URL: <https://github.com/>.
- [15] Mark Haakman et al. “AI lifecycle models need to be revised”. In: *Empirical Software Engineering* 26 (Sept. 2021), p. 95. DOI: [10.1007/s10664-021-09993-1](https://doi.org/10.1007/s10664-021-09993-1).
- [16] Samridhi Jain and Puneet Kumar. “Cost Effective Generic Machine Learning Operation: A Case Study”. In: *2023 International Conference on Data Science and Network Security (ICDSNS)*. 2023, pp. 1–6. DOI: [10.1109/ICDSNS58469.2023.10245408](https://doi.org/10.1109/ICDSNS58469.2023.10245408).
- [17] Meenu Mary John, Helena Holmström Olsson, and Jan Bosch. “Towards MLOps: A Framework and Maturity Model”. In: *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2021, pp. 1–8. DOI: [10.1109/SEAA53835.2021.00050](https://doi.org/10.1109/SEAA53835.2021.00050).
- [18] Justin Domingus John Arundel. *Cloud Native DevOps with Kubernetes*. O’Reilly Media, 2019. ISBN: 9781492040712.
- [19] Dominik Kreuzberger, Niklas Kühn, and Sebastian Hirschl. “Machine Learning Operations (MLOps): Overview, Definition, and Architecture”. In: *IEEE Access* 11 (2022), pp. 31866–31879. URL: <https://api.semanticscholar.org/CorpusID:248524628>.
- [20] *Kubeflow*. URL: <https://www.kubeflow.org/>.
- [21] *Kubernetes*. URL: <https://kubernetes.io/docs/concepts/>.
- [22] Bo Li et al. “Trustworthy AI: From Principles to Practices”. In: *ACM Comput. Surv.* 55.9 (Jan. 2023). ISSN: 0360-0300. DOI: [10.1145/3555803](https://doi.org/10.1145/3555803). URL: <https://doi.org/10.1145/3555803>.
- [23] Yan Liu et al. “Building A Platform for Machine Learning Operations from Open Source Frameworks”. In: *IFAC-PapersOnLine* 53.5 (2020). 3rd IFAC Workshop on Cyber-Physical Human Systems CPHS 2020, pp. 704–709. ISSN: 2405-8963. DOI: <https://doi.org/10.1016/j.ifacol.2021.04.161>. URL: <https://www.sciencedirect.com/science/article/pii/S2405896321003013>.
- [24] Sasu Mäkinen et al. “Who Needs MLOps: What Data Scientists Seek to Accomplish and How Can MLOps Help?” In: *CoRR* abs/2103.08942 (2021). arXiv: [2103.08942](https://arxiv.org/abs/2103.08942). URL: <https://arxiv.org/abs/2103.08942>.

- [25] “MLOps: Continuous delivery and automation pipelines in machine learning”. In: (). URL: <https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning>.
- [26] Andrei Paleyes, Raoul-Gabriel Urma, and Neil D. Lawrence. “Challenges in Deploying Machine Learning: A Survey of Case Studies”. In: *ACM Comput. Surv.* 55.6 (Dec. 2022). ISSN: 0360-0300. DOI: [10.1145/3533378](https://doi.org/10.1145/3533378). URL: <https://doi.org/10.1145/3533378>.
- [27] Deven Panchal et al. “MLOps: Automatic, Zero-Touch and Reusable Machine Learning Training and Serving Pipelines”. In: *2023 IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS)*. 2023, pp. 175–181. DOI: [10.1109/IoTaIS60147.2023.10346079](https://doi.org/10.1109/IoTaIS60147.2023.10346079).
- [28] Deven Panchal et al. “Reusable MLOps: Reusable Deployment, Reusable Infrastructure and Hot-Swappable AI models and services”. In: *2024 10th International Conference on Smart Computing and Communication (ICSCC)*. 2024, pp. 645–651. DOI: [10.1109/ICSCC62041.2024.10690392](https://doi.org/10.1109/ICSCC62041.2024.10690392).
- [29] Nenad Petrovic. “Model-Driven Approach to Blockchain-Enabled MLOps”. In: June 2022.
- [30] Nataša Radaković, Ivana Senk, and Nina Romanić. “A MACHINE LEARNING PIPELINE IMPLEMENTATION USING MLOPS AND GITOPS PRINCIPLES”. In: Jan. 2023, pp. 94–99. DOI: [10.24867/IS-2023-T2.1-6_08141](https://doi.org/10.24867/IS-2023-T2.1-6_08141).
- [31] Aiswarya Raj et al. *From Ad-Hoc Data Analytics to DataOps*. July 2020. DOI: [10.13140/RG.2.2.36807.93604](https://doi.org/10.13140/RG.2.2.36807.93604).
- [32] RedHat. “What is a GitOps workflow”. In: (2024). URL: <https://www.redhat.com/en/topics/devops/what-is-gitops-workflow>.
- [33] Miguel G. Rodrigues et al. “A MLOps architecture for near real-time distributed Stream Learning operation deployment”. In: *Journal of Network and Computer Applications* (2025), p. 104169. ISSN: 1084-8045. DOI: <https://doi.org/10.1016/j.jnca.2025.104169>. URL: <https://www.sciencedirect.com/science/article/pii/S1084804525000669>.
- [34] Gema Rodriguez-Perez et al. “A unifying framework for the systematic analysis of Git workflows”. In: *Information and Software Technology* 138 (2021), p. 106618. DOI: [10.1016/j.infsof.2021.106618](https://doi.org/10.1016/j.infsof.2021.106618).
- [35] Nirmal Sajanraj. “DataOps and MLOps: Implementation Patterns across Industries”. In: *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* 11 (Mar. 2025), pp. 1545–1554. DOI: [10.32628/CSEIT25112697](https://doi.org/10.32628/CSEIT25112697).
- [36] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. “Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices”. In: *IEEE Access* 5 (2017), pp. 3909–3943. DOI: [10.1109/ACCESS.2017.2685649](https://doi.org/10.1109/ACCESS.2017.2685649).
- [37] Jasper Stone et al. *Navigating MLOps: Insights into Maturity, Lifecycle, Tools, and Careers*. Mar. 2025. DOI: [10.48550/arXiv.2503.15577](https://doi.org/10.48550/arXiv.2503.15577).

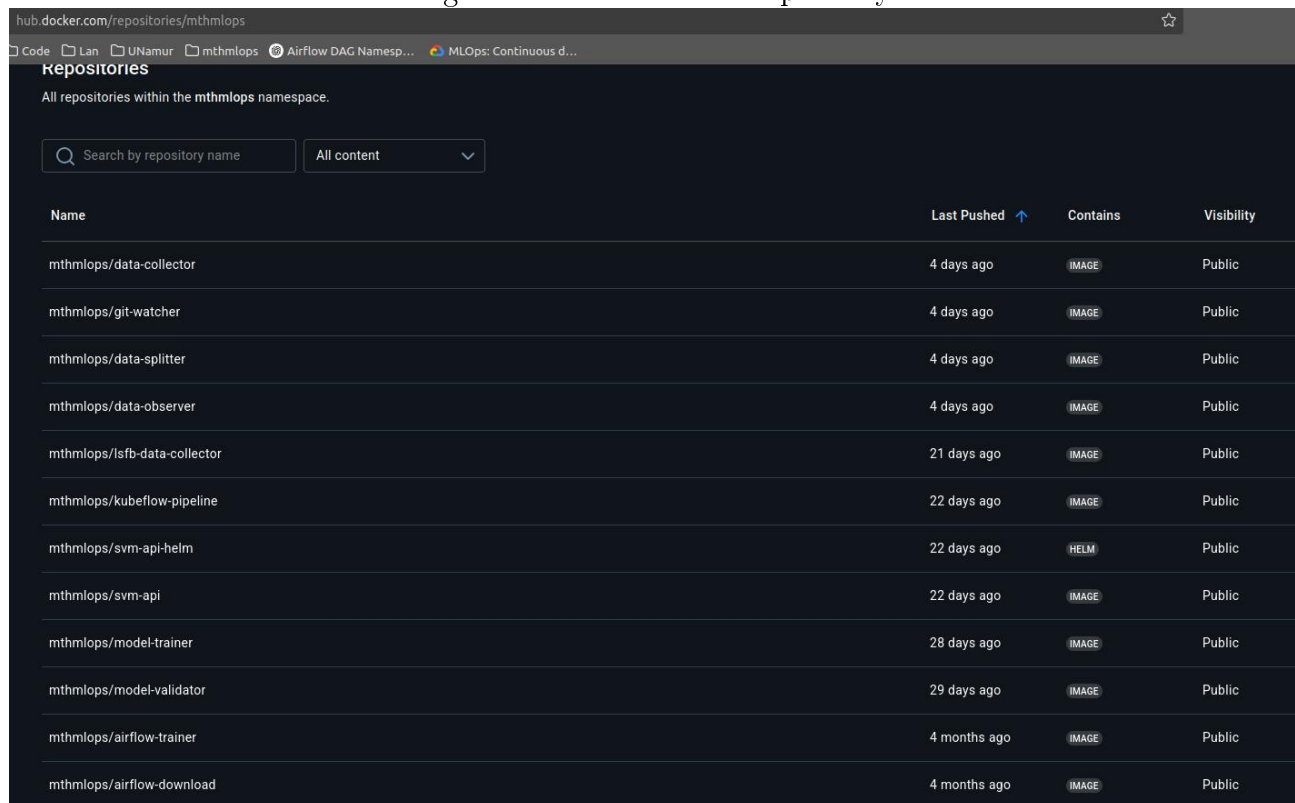
- [38] Georgios Symeonidis et al. “MLOps - Definitions, Tools and Challenges”. In: *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*. 2022, pp. 0453–0460. DOI: [10.1109/CCWC54503.2022.9720902](https://doi.org/10.1109/CCWC54503.2022.9720902).
- [39] Matteo Testi et al. “MLOps: A Taxonomy and a Methodology”. In: *IEEE Access* 10 (2022), pp. 63606–63618. DOI: [10.1109/ACCESS.2022.3181730](https://doi.org/10.1109/ACCESS.2022.3181730).
- [40] M. Treveil et al. *Introducing MLOps*. O’Reilly Media, 2020. ISBN: 9781098116446. URL: <https://books.google.be/books?id=BioMEAAQBAJ>.
- [41] “what is devops”. In: (). URL: <https://github.com/resources/articles/devops/what-is-devops>.
- [42] Adrian P. Woźniak, Mateusz Milczarek, and Joanna Woźniak. “MLOps Components, Tools, Process, and Metrics: A Systematic Literature Review”. In: *IEEE Access* 13 (2025), pp. 22166–22175. DOI: [10.1109/ACCESS.2025.3534990](https://doi.org/10.1109/ACCESS.2025.3534990).

Appendix

Below are the GitHub and DockerHub repositories related to this project:

- **Infrastructure:** <https://github.com/Baylib/mthmlops-infra>
- **Airflow and Kubeflow pipelines:** <https://github.com/Baylib/airflow-dags>
- **Model Api:** <https://github.com/Baylib/svm-api>
- **Image and HelmChart repository:** <https://hub.docker.com/repositories/mthmlops>

Figure 2.16: DockerHub repository



hub.docker.com/repositories/mthmlops

repositories

All repositories within the mthmlops namespace.

Search by repository name All content

Name	Last Pushed ↑	Contains	Visibility
mthmlops/data-collector	4 days ago	IMAGE	Public
mthmlops/git-watcher	4 days ago	IMAGE	Public
mthmlops/data-splitter	4 days ago	IMAGE	Public
mthmlops/data-observer	4 days ago	IMAGE	Public
mthmlops/isfb-data-collector	21 days ago	IMAGE	Public
mthmlops/kubeflow-pipeline	22 days ago	IMAGE	Public
mthmlops/svm-api-helm	22 days ago	HELM	Public
mthmlops/svm-api	22 days ago	IMAGE	Public
mthmlops/model-trainer	28 days ago	IMAGE	Public
mthmlops/model-validator	29 days ago	IMAGE	Public
mthmlops/airflow-trainer	4 months ago	IMAGE	Public
mthmlops/airflow-download	4 months ago	IMAGE	Public