Deep Science Publishing
https://doi.org/10.70593/978-93-49910-08-9_3

DeepScience
Open Access Books

Chapter 3: Designing and deploying scalable MLOps pipelines for continuous artificial intelligence model training and delivery

3.1. Introduction

Artificial Intelligence (AI) systems are rapidly being adopted across industries to address important business use cases. Healthcare, financial services, and public safety organizations are increasingly integrating AI into their operations to better enhance decision making, improve business processes, provide superior customer service, ensure compliance to regulations, and reduce costs. AI systems are impacting real business processes; improving early-stage diagnosis of various diseases to ensure timely preventive care, assisting financial advisors to recommend tailored investment plans, predicting violations of laws and rules on social media to enable timely intervention, and helping major retailers optimize their supply chains and improve delivery efficiency. These advantages are resulting in the adoption of AI for mission critical tasks where the cost of failure for the business could be extremely large. For such high stake business applications, the level of explainability and interpretability required from AI systems becomes very high since these businesses are risking core trust and relationship with end customers that have been built over many years (Liang et al., 2024; Mallardi et al., 2024; Jain & Das, 2025).

Due to their high complexity, AI systems cannot be designed, developed, and deployed by a single organization or a small team within an organization. Consequently, AI systems are designed and built across different functional teams within an organization and most often across organizations. Within an organization, different teams are responsible for adhering to specific business objectives around success factors such as accuracy, latency, and prediction pricing while enabling AI services to orchestrate relevant workflows across various teams and data and infrastructure services to integrate

data and compute resources at scale. This leads to the rise of a large number of assets, along different stages of an ML's life cycle both in isolation and at scale. All these assets need to be managed holistically in a systematic manner to enable enterprise grade ML delivery which involves an integrated and cohesive process that focuses on training AI and ML models and delivering AI-powered business applications that satisfy preferred business objectives, explainability and interpretability constraints, and accuracy, latency, and agility requirements while managing business risks (Matthew, 2022; Singla, 2023; Slade, 2024).

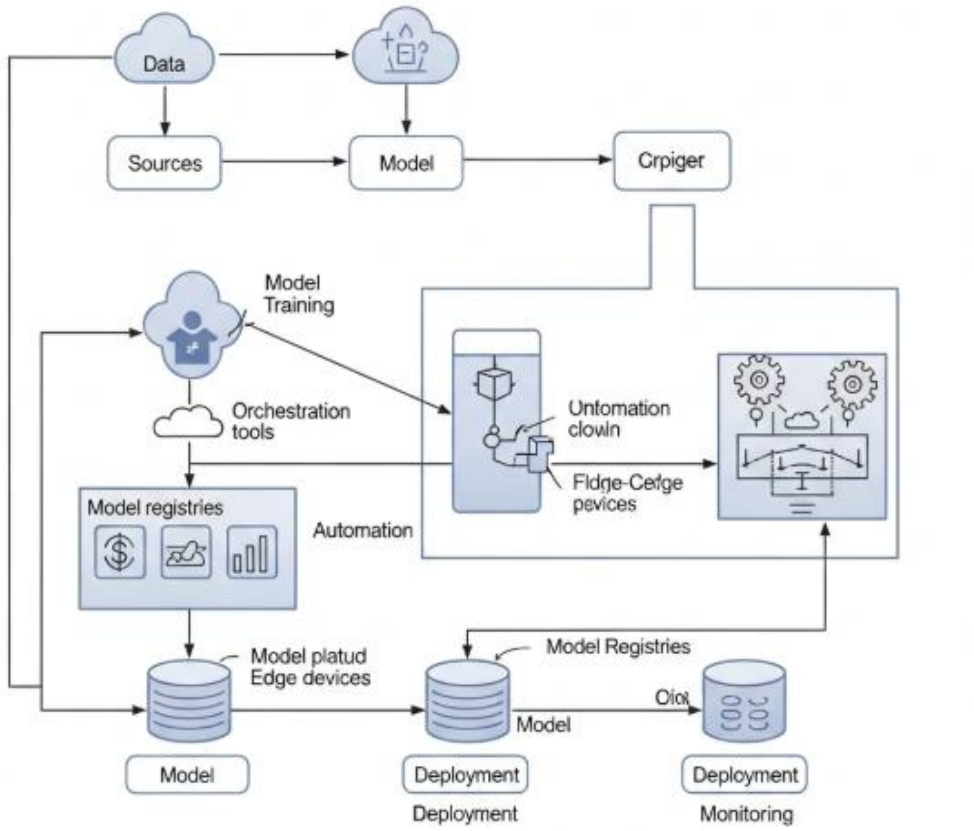

Fig 3.1: Designing and Deploying Scalable MLOps Pipelines

3.1.1. Background and Significance

MLOps is a collaborative approach for Data Science, IT operations, production engineering, and software development teams to manage AI solutions. MLOps automates the AI solution development and delivery process, like traditional DevOps, enhancing continuous delivery and release management practices. Diverse technologies sum to MLOps solution pipelines: Continuous integration - CI (tools that facilitate the

merging of developer changes into a central repository ensuring that deployed code does not break downstream processes); Data versioning (tools that keep a consistent and up-to-date catalog of datasets); Data testing (tools that validate dataset coherence and integrity); Model versioning (tools that keep a consistent and up-to-date catalog of models); Model testing (tools that validate the performance and behavior of AI solutions); Model deployment (tools that push new models into a deployable production); Continuous delivery - CD (a collection of best practices that enable teams to release code in shorter cycles and with less risk).

MLOps align with the goals of organizations that successfully democratize data and AI within their boundaries. These companies realize substantial business value from their AI initiatives since they easily and frequently replace their (most times) brittle early model versions with automated retraining. These replacement cycles can be tiring for Data Science and IT ops teams involved in the MLOps activity, but a careless approach can impact the performance and behavior of deployed models to the point of risk to the companies. Issues such as the retrained model being unable to perform as well as its predecessors are common. Other issues may derive from model bias, e.g., a classifier model for registering governance issues that retracts previous classifications after retraining.

3.2. Understanding MLOps

MLOps, or Machine Learning Operations, is a set of practices that combine Machine Learning, DevOps, and Data Engineering to deploy and maintain ML models in production reliably and efficiently. MLOps aims to automate the deployment, monitoring, and management of ML models in production environments. It is an important part of the Machine Learning lifecycle, particularly the later stages. There is a growing demand for deploying Machine Learning solutions in enterprises, and MLOps is critical to advancing this field. Maturity of DevOps teams within enterprises can have a positive impact on the success of MLOps initiatives. MLOps is a collaborative function, often comprising data scientists, devops engineers, and data engineers. MLOps aims to close the gap between Machine Learning models in development and Machine Learning models in production. The process of taking a model into production can be brittle and complicated, especially when an organization is building multiple models for various internal and external customers. These multiple models may have multiple definitions of success, and different stakeholders across the organization may be involved in the development of these models. Automating various components of the infrastructure for deploying Machine Learning models can ensure that the models are production-ready and that the time spent in the iterations of model training and data validation is drastically reduced. This, in turn, helps in quickly realizing the benefits of

deploying Machine Learning models within enterprises. While there are tools to optimize various components of the MLOps architecture, organizations still need a systematic design approach to articulate the right framework.

3.2.1. Definition and Importance

Machine Learning Operations (also referred to as MLOps) is a practice that aims to simplify the tasks of continuous monitoring, maintenance, and delivery of AI models into production at scale. MLOps "encompasses the diverse roles, volumes, velocity, and veracity of data at all stages of ML development and deployment, focusing on the operational aspects of continually delivering production-grade ML systems." Specifically, MLOps enables the development team, such as data scientists and ML engineers, to collaborate with IT teams, including DevOps and SRE, to ensure and manage the workflow, storage, and orchestration of data processing, training models, testing and validating models, checking and monitoring models in production, and model retraining and redeployment. MLOps specializes in seamless integration and support for the aforementioned steps, with a strong emphasis on model reproducibility.

The main emphasis of MLOps is on speeding up the development of ML pipelines and easing operational burdens on ML engineers and developers. MLOps provides declarative interfaces for easy specification and setup of ML workflows and pipelines, promoting data integrity and tracking of workflow execution and storage, ease of integration of various ML and component tools, and model version control. Overcoming these operational obstacles leads to multiple benefits from reducing time in handover between data scientists to deploy the model in production and engineers, reducing ML experiment duplication, better resource utilization from cluster orchestration and permission management, which can lead to quicker turnarounds in model inference latency and availability. Ultimately, MLOps helps scale down the problem of the continuous deployment of specialized ML pipelines as is commonly done in the software discipline for regular software development, testing, and deployment.

3.2.2. Key Components of MLOps

MLOps is a relatively nascent concept, and several companies are trying to provide solutions for providing MLOps systems. Each solution is unique, and, consequently, both the flexibility and complexity vary widely over these solutions. Although there is no specific area of MLOps where every solution must excel, every MLOps product needs to integrate several key MLOps functions and tasks seamlessly to be effective and useful to its end users. In this section, we will describe some of the key components of MLOps needed to allow for a seamless integration of the MLOps tasks.

At the core of MLOps is effective communication and collaboration between multiple users and components of a machine learning lifecycle. A typical ML project is not the work of a single scientist. Often multiple people at various stages of the project are working on and off on the project, and hence a proper MLOps pipeline should have some sort of project dashboard to help keep track of the state of the models and stage of the dependencies.

3.3. Architectural Considerations

An MLOps platform by definition supports training and delivery of ML models in a scalable and automated manner. But what does scalability mean? There are several points at which we could ask this question. The most general meaning is that we can add more capacity, in the form of more clusters or machines and our system can do more work, either increasing the speed at which you may deploy a model or increasing the number of different models you can deploy. Secondly, most training tasks are no longer single-threaded processes, so the amount of work that we are doing in a single task may be very large, yet throwing more machines at the problem can only help so much. Thirdly, even distributing training is a limited operation. How many types of ML tasks can we do at once? Are they different in terms of the resources we use compared to the batch inference process during production time? Can these resources be assigned manually or do we need to manage these resources somehow?

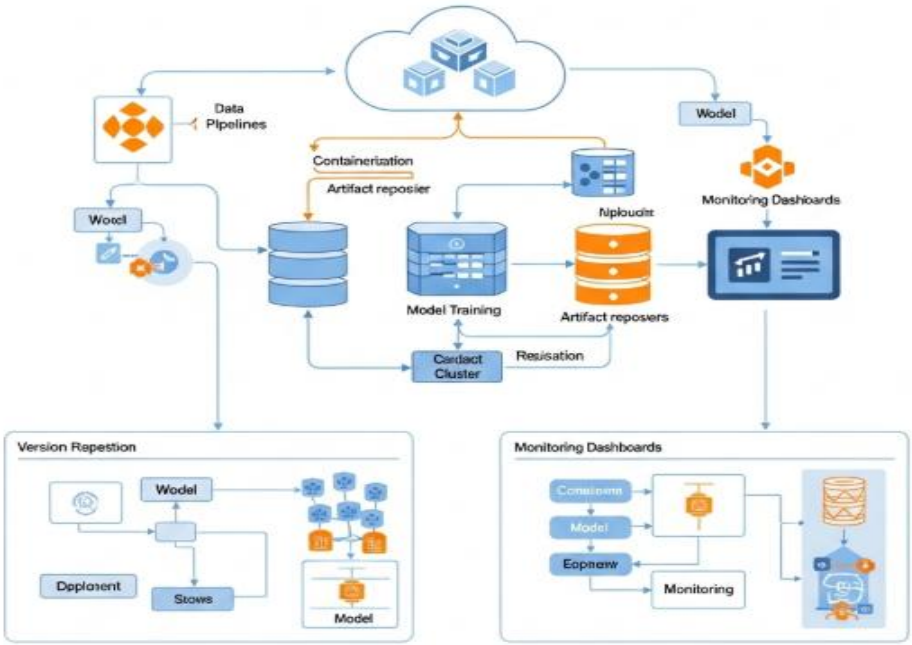

Fig 3.2: Architectural Considerations

Here, we define a few specific use-cases that an MLOps platform would need to support. One or more of them would need to be supported for a proper MLOps type of workflow to be considered. These tasks are not independent of each other, and in a properly designed architecture, support of one feature should not result in the negation of other pipelines. As such, we can have batch ML tasks running while real-time pipelines are active, or schedule larger batch jobs in the background. Systems resource management for such a system needs to factor both data and model complexity together while managing state appropriately during model training for an end-to-end system that supports both real-time serving and batch processing of predictions.

3.3.1. Scalability Requirements

In the previous chapter we observed that not only is the training of AI ML and deep learning models an essential part of delivering AI-powered applications, it is also a highly complex process that involves many engineering disciplines, such as data engineering, application development, systems engineering, enterprise architecture, and security. The next logical step is to understand the scalability requirements for MLOps pipelines, which must scale as the AI training problems and the enterprise that they operate in grow.

Data. The enterprise and its business operations generate enormous amounts of diverse data: transaction logs, sensor data, social media data, and many more. This enterprise data is often enriched with third-party external datasets to build and train AI models. The size and dimensionality of these AI datasets may vary with different business use cases and over time, but they will continue to grow. Datasets with billions or even trillions of data points and thousands of features are not uncommon.

Model. The data used to train AI ML and deep learning models continually grow in volume. The number of AI models deployed in production also continues to grow as well. These models become more complex over time as they are enhanced with new features and larger, more complex architectures. These models may become very deep, as in the case of deep learning neural networks with over billion parameters, and can require enormous amounts of training dataset to be accurately modeled and deployed. These models also have multi-million parameter architectures that require months to train on dozens of cutting-edge GPUs and TPUs. These requirements are only expected to grow in the future.

3.3.2. Microservices vs. Monolithic Architectures

Key challenges in automating continuous training for AI models at scale revolve around orchestration, scheduling, and resource management. For example, should a team adopt a microservices architecture, or is a monolithic one sufficient? What are the trade-offs between an orchestration, resource management, and scheduling approach that builds on top of existing data plane services? What are the key centralized points of failure and contention when sharing infrastructure resources across multiple machine learning teams?

Microservices architectures are commonly recommended for cloud-based applications. In such architectures, each service is decoupled and organized into distinct services, with well-defined APIs for communication. Each microservice can be operated, deployed, and scaled independently. Scaling is possible not only horizontally, deploying multiple instances, but also vertically, allocating more hardware resources. The many benefits, such as reduced time to market and fast and effective security audits, can also be gained through a monolithic architecture. In fact, arguably, at small scale, the flexibility and simplicity associated with the monolithic approach outweigh the overhead associated with a microservices architecture.

The monolith has been a widely adopted web app structure since the earliest days of the Internet, and, for notable startups, allowed developers to go rapidly from unconscionable technical debt to exit. However, for medium to large deployments, the architecture becomes increasingly harder to reason about. Architectural components that were originally decoupled and independent of each other become intertwined and more difficult to update, scale, and manage. Due to its success, almost every modern web infrastructure began its life as a monolith. Once a company scales its user base to millions, the organizations migrate to a microservices architecture. They break up their monolith into small pieces, which are easier to manage and scale independently.

3.4. Data Management Strategies

Data fuels every machine learning (ML) model process, from training to serving. In MLOps, maintaining the quality, integrity, and storage of the underlying data adds challenges, but these are important considerations for successfully deploying production ML services. As part of the MLOps pipeline, data management builds off of general data engineering techniques but adds nuance and features to support ML initiatives like data drift monitoring and ensuring data quality for ML tasks.

Data Collection and Storage

The first step in maintaining data in MLOps is to acquire data and do the necessary preprocessing steps to put it in a form usable for a model. This may involve scraping data, downloading images, or getting sensor data from an IoT device. These assets typically live on filesystems, cloud storage services, or databases. Allocating storage plans, and optimizing retrieval functions and times, is important. New datasets may need to be collected from the original source every so often or on-demand for model inference. The retrieved and processed data may then need to be stored until the next usage as training data, either in an intermediate or final storage system.

Data Versioning and Lineage

Big data nowadays can mean terabytes of information. Storing, processing, and accessing this data — at the right time and in the right form — require careful consideration, especially when one ML model retrains on that data for maintenance, freshness, and improvement. ML models are sensitive to changes over time: the underlying training and inference data will have notable differences from when the model was first deployed.

3.4.1. Data Collection and Storage

With increasing data volumes, operationalizing MLOps requires planning for multiple data management strategies, from data collection and storage to data versioning and lineage. In this section, we discuss various data strategies necessary in practice, some of which should be on auto-pilot, delegated to data engineering capabilities.

How should data be collected and how should it be stored? To answer these questions, we consider various aspects involving data velocity, variety, and volume. MLOps may be set up to utilize data across a large variety of sources, from custom crewmember and aircraft data to third party data concerning weather, location, and events nearby, to basic telemetry data from the aircrafts, plus augmented data pipelines to create composite time series data.

Data volume can significantly affect decision-making in terms of data storage, as well as systems resources and costs. Both have led to a recent trend towards using exclusively streaming data in model training; micro-batch or one-second data latency using the most recent past data; plus multiple semantic-aware composites of observed data, replacing historical data stored on cheaper cloud-based data lakes, due to limited cost-effective use of historical learnings enabled by batch processing in this regime.

3.4.2. Data Versioning and Lineage

In the previous chapter, we saw that Machine Learning Ops pipelines (MLOps pipelines) processes data that need to be collected and stored somewhere. It is useful to organize this data for easy access while it is worked on, especially those datasets used to train and test production ML models. It is also important to keep track of changes as the ML training data will evolve as the product is developed, since the relationship of effects and causes on the product's input-output variables may not be clear at first. The model training itself may affect the product's data, and the product's data affect the model's training. Having a clear way to identify the data version and the lineage of each dataset is an important part of the data management strategy of an MLOps pipeline. In this section, we will explore these two closely related tasks, what they are, why they are useful, their higher-level intention, and options that the tools and frameworks used for managed data storage offer.

Any data collected and stored by a company or organization will evolve as more data is captured, but they may also be cleaned and curated to better represent a company's objective before it is used to train ML models. These changes will affect the representation of the ML pipeline input-output relationship and affect the pipeline output variables. Being able to track which dataset version was used in the training process of each model version is an indispensable resource for interpreting the results of each model in production, and a pre-requisite for any model retraining task. Having a detailed log of prior data versions is also a good starting point for improving the input-output mapping. Sometimes an ML model's worse-than-expected performance can be traced to a simple problem in the data quality. A good practice when collecting data from production systems is to keep the previous versions of the data stored somewhere.

3.5. Model Development Lifecycle

After addressing the problem formulation and data collection steps, it's time to prepare your data so you can begin the actual AI model training. You want to make sure you prune your datasets to focus on your core objective defined in the problem statement. In addition to the basic granularity of your data, there are many other considerations you might take into account based on the AI objective you're trying to achieve: the ratio of different classes should be considered and might require upsampling of the minority classes in cases of classification problems; for time-series data you should also take into account seasonality effects; or in cases of generative models you might want to define a hierarchy of attributes to make it easier to train your model. It's not necessarily a straightforward process. Each of these applications is unique to different models, specifically tuned by applying past knowledge and research on the specific type of model you are inventing.

Once the datasets have been prepared, you are ready to start training your models. Most of the time, model training will not only consist of a single step. It's usually a multistep process or hierarchy that requires performing multiple steps involving many techniques. At a high level, it usually consists of two types of operations: the actual training, and hyperparameter tuning to improve the training loss so that the model reduces the overall bias and variance potential before running on the actual testing and validation sets. The actual training usually consists of using a model with default hyperparameter values trained on the available datasets with a few optimization heuristics applied to it.

3.5.1. Model Training Techniques

The model training is the process of adjusting the model weights to perform a given task, such as classifying input data, predicting a value, or generating a new input. Model training methods vary depending on the type of machine learning task and algorithm used. We will cover the following types of training techniques.

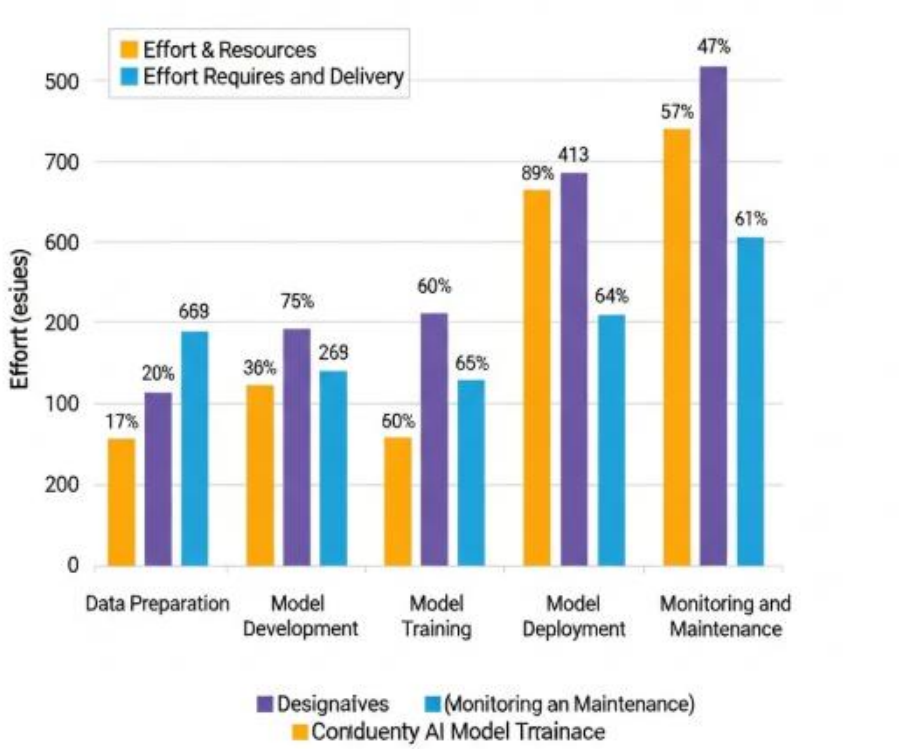

Fig : MLOps Pipelines for Continuous AI Model Training and Delivery

Supervised Learning. In supervised learning, we train the model using already labeled data. The label is the desired prediction or value of the ground-truth that the machine

learning system needs to predict or contribute in order to be useful. The training data contain the input features and the label for each sample. In particular, Federated Learning is a distributed method of supervised training in which multiple devices calculate the gradients of the model using their own private datasets, and only the updates are sent to the coordinating server. The global model is then updated based on these updates instead of using the sample inputs, preserving privacy. It is a very popular training method in personal devices to create smarter models without exposing private data to data-hungry AI service providers.

Unsupervised Learning. In unsupervised learning, we train the model using unlabelled data. The goal of the training process is to find clusters or structures in the dataset, such as groups of customers with the same purchasing habits, formatting a search query, or those images with the same features. The Infomax model uses an unsupervised approach for contrastive learning of visual representations. It operates by maximizing the info of negative samples generated at training time based on the original data without labels to disentangle the visual appearance.

3.5.2. Hyperparameter Tuning

Hyperparameters control the behavior and performance of machine learning models. Hyperparameter tuning validates a model's hyperparameter settings to configure them to maximize model performance. Compared to model training – where model parameters learn from training data – hyperparameter tuning is up via a validation set and not part of training. An appropriate search algorithm examines the search space of hyperparameter configurations and a performance metric determines the level of success during the search. Tuning is difficult, capturing much of the art versus science in ML model development.

An isolated ML model development effort – where just one final model is trained on a full dataset – usually employs grid search or random search. Hyperparameter tuning for production-ready pipelines, however, is not a one-shot effort. An initial coarse tuning is often first performed to select promising hyperparameter regions, which are then used in a finer search for final tuning. It is vital for pipelines that retrain models with regularity to also include the costs of tuning, especially for distributed batch pipelines with a single model retraining at a time. As such, parallelism, budget, scheduling, dependency, and performance considerations are foundational for pipelines performing hyperparameter tuning on production models. Those considerations impact both optimizing a single model and concurrent runs of pipeline components performing hyperparameter tuning for different models, including spatial parallelism and other effects, such as non schedulable budgets where budgets are defined for a set of scheduled runs and not for individual runs.

3.6. Continuous Integration and Continuous Deployment (CI/CD) for ML

CI/CD enables practices and tools for automating how applications and services are built, tested, validated, and deployed to production environments. These development pipelines reduce the time and effort it takes to integrate updates and changes to existing systems and enable rapid updates and delivery for existing products and services.

Readily automatable steps in the overall model development lifecycle can follow already-edified software CI/CD practices and tools. Automated testing and deployment processes are crucial to enabling rapid development cycles for production ML applications. However, CI/CD tools that apply to typical software projects often fail to support the unique characteristics of ML - where the code has no control over the outcome and where systems with different code and model versions interact with each other even after deployment.

A sizable portion of developing and releasing a "good" model involves diagnosing when and how to anticipate, monitor, and mitigate model performance degradation. However, many of the analysis steps that practitioners perform to assess model performance are currently manual and undocumented - adding significant amounts of effort, time, and risk to deploying and monitoring ML models. Efforts are ongoing within the ML community about developing model performance measurement protocols and recommending end-to-end MLOps pipelines with testing workflows specifically for ML.

3.6.1. CI/CD Pipelines for ML Models

The CI/CD Pipelines for ML Models is the section of the Content that explains how CI/CD can be extended to support ML and to add not just testing for correctness but additional tests that validate and check if results are still acceptable and within expected norms. We also briefly describe use of Jenkins for running these CI/CD pipelines for MLOps and AI model development and deployment. Continuous Integration and Continuous Deployment has been used in Enterprise IT for many years now. It was recognized pretty early that in software, code and deployment could be automated enough and the time required to put production-ready code was short enough that these software development lifecycle phases could be automated and made agile.

MLOps of course cannot be quite as agile. The challenges in model development (feature, labels, model training, etc.) are very real and so requisite to make the CI/CD pipelines for ML Models that CI/CD check both common IT concerns and ML concerns. There are two CI/CD Pipeline tasks and phases that we will cover in this section. The first is the execution or calling of pipelines to automate and execute stages of development and production deployment and lifecycle phases such as Featurization, Model Training and Training Validation, Model Quality Assessment, Model

Deployment and Input Injecting and Inference Collection, and Actual Inference and Predictions. These are not strictly the Pipeline tasks that can do nor do they necessarily have to exist in CI/CD Containers. These can include calls to Data and ML services and functions. Some of these functions can be called from CI/CD Pipeline calls to simple functions or scripts based on execution time or function layer metrics checking and alerts.

3.6.2. Automating Model Testing

Model testing is performed to ensure that a trained model is desirable and that the results that it provides are reasonable. The predictions that a model provides depend on the input data. Model testing consists of two components: data validation and model testing. Automated data validation should be performed to ensure that there is no data drift from previous model inputs and to catch data errors. Another option for data validation is the use of the Model Monitor.

Model testing should check the model's predictions and their properties. This includes checking for prediction drift across time, prediction dependency, and prediction safety. Prediction drift occurs when the distributions of the predictions change over time, and there are quota or frequency thresholds that govern the safety of prediction drift. Prediction dependency occurs when the predictions by the model change when certain feature variables still have the same input value. For example, if certain images always return the same infringing value, then the model is usable for that case. Prediction safety occurs when the predictions of a problem can be caused by the presence of certain input features. A trained model should be tested for this condition to ensure that the predictions are not harmful to society.

Automated model testing is possible. There are some publicly available tools that you can use. As well, internal, custom models and techniques are also possible for testing as well. Model testing is also generally incorporated into the training and CI/CD pipelines, for specialized handles. Model testing acts as a guide for retraining the model when needed.

3.7. Conclusion

The combined design and deployment for scalable MLOps pipelines and their solutions addresses the challenges inherent in deploying, monitoring, and retraining AI models with adaptive embedded logic and format, semantic, and temporal drift with minimal human intervention. By relying on humans only to prepare the required domain knowledge, these pipelines enable domain leads to provide input for faster and easier

validation rather than engineers needing to develop and integrate the models, systems, and pipelines required for deployment. Effectively encapsulating the model code and application with parameterized wrappers makes it easy to develop and deploy new models that use the same logic or the same model, with different logic for different portions of the data space without requiring significant changes to the model code or redeployment of the development infrastructure. Parameterization also enables rapid optimization of the model parameters, centrally for model types like ML code and NLP models with external parameter customization that affect all model instances. Predefined central pipelines that include the AI model deploy and retraining and delivery process allow for continuous operation of pipelines, triggered at the end of each run of the base pipelines by user-defined metrics that indicate the need to trigger the customizable events contained in the event pipeline.

We expect that further advancements in pipeline design, along with the continued adoption of MLOps by organizations, will lead to revolutionary advances in ML and NLP, further reducing the need for domain specialists, who will instead only need to monitor semantic-based metrics for drift detection to make decisions on changes to the required domain input. Domain leads will also need to specify pipelines for tasks that differ from, or are hybrids of previously specified and implemented pipelines. For more general classes of tasks, further exploration of the use of closed AI pipelines will lead to significant accuracy improvements as both domain actants and technical reference models and generative nodes become increasingly intelligent.

3.7.1. Future Trends

As companies worldwide sustain unprecedented demand for AI-powered tools and services, it is hard to perceive a slowdown in the deployment of modern AI technologies. Natural language processing is in a constant upward trajectory, growing at warp speed with advances in generative pre-trained transformers for natural understanding and learning with few examples. Powerful foundation models trained at massive scale for specific purposes will unleash capabilities that could propel industries into a new dimension. The convergence of generative models in longview vision, synthetic video generation, 3D modeling, large language-driven robotics, and others will require new interactions with the world. Metaverse as a service requires compute power combined with seamless introduction of simulation, real-time capabilities, and large global interactions, all built on collaborative engines focused on global challenges.

Fostering responsible AI, which reduces biases and reflects a diverse representation of the world to avoid AI hallucinations, while accountable AI, which guarantees compliance, privacy, cyber threat deterrence, and control are among the crucial elements in ensuring such trust. New regulatory policies and funding incentives will permeate the

business landscape. Democratizing state-of-the-art AI tools and resources faster, more efficiently, and proactively for historically excluded communities and individuals will improve the ecosystem's knowledge base and empower a more inclusive labor market. AI will evolve to guarantee that every individual has the tools to be successful, which will become one of the most important pieces in building a more sustainable knowledge economy.

References

Matthew, B. (2022). MLOps and DataOps Integration: The Future of Scalable Machine Learning Deployment.

Jain, S., & Das, J. (2025). Integrating data engineering and MLOps for scalable and resilient machine learning pipelines: frameworks, challenges, and future trends.

Slade, R. (2024). Adaptive MLOps Strategies for Scalable and Sustainable AI Deployments.

Singla, A. (2023). Machine learning operations (mlops): Challenges and strategies. Journal of Knowledge Learning and Science Technology ISSN: 2959-6386 (online), 2(3), 333-340.

Liang, P., Song, B., Zhan, X., Chen, Z., & Yuan, J. (2024). Automating the training and deployment of models in MLOps by integrating systems with machine learning. arXiv preprint arXiv:2405.09819.

Mallardi, G., Calefato, F., Quaranta, L., & Lanubile, F. (2024, December). An MLOps Approach for Deploying Machine Learning Models in Healthcare Systems. In 2024 IEEE International Conference on Bioinformatics and Biomedicine (BIBM) (pp. 6832-6837). IEEE.