IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)
An International Open Access, Peer-reviewed, Refereed Journal

Machine Learning Integration In CI/CD Pipelines: Challenges, Architectures, And Case Study Evaluation

Kavya Dhingra
Department of Computer Science and Technology
Manav Rachna University
Faridabad,India

Prisha Chopra
Department of Computer Science and Technology
Manav Rachna University
Faridabad,India

Nikita Sharma
Department of Computer Science and Technology
Manav Rachna University
Faridabad,India

Dr.Meena Chaudhary
Department of Computer Science and Technology
Manav Rachna University
Faridabad,India

Dr.Narender Gautam
Department of Computer Science and Technology
Manav Rachna University
Faridabad,India

Abstract Using Machine Learning in CI/CD pipelines brings about specific challenges since models depend on data (data can vary), they evolve over time during the training/exploitation stage (initially a single model is produced for deployment/testing), and the model may require continued training and/or retraining. CI/CD pipelines as they exist today are meant for deterministic code (e.g., non-ML libraries) and do not take into consideration additional important ML system metrics such as data drift and model performance. In this paper, we summarize and highlight the major issues for ML CI/CD implementation, we describe some concrete architecture patterns to deal with identified issues, and give a case study using GitHub Actions and Azure Machine Learning. Results on evaluation show that MLOps based CI/CD has increased ability to release more rapidly, the full pipeline is more reliable, and operationalizing bad models can be avoided.

Introduction

Machine Learning is increasingly becoming a significant aspect of real-world software systems. In contrast to traditional software, ML models are not permanent, are always dependent on data and usually degrade in performance with any context changes. Therefore, the iteration in a typical continuous CI/CD pipeline utilized in engineering does not fit into the

framework of ML. MLOps makes up many of the development, deployment, and monitoring steps in the management of ML and may include dataset verifications, experiment tracking, metrics based validation of the models, and packaging of the processes within containerized parameters for monitoring of the run-time processes. In essence, MLOps is a melding of the best practices learned from DevOps with the best practices for managing the ML life cycle.

Accordingly, CI/CD pipelines that are being utilized organizationally with a ML process, essentially, automates the building, evaluating, and deployment functions of the models. Not only does this create a reduction in the manual labour required for these processes, but it also allows for the best models to get into production in a shortened timeframe. This article will explore the challenges of applying ML to CI/CD pipeline use in production ML reference architecture and an industry case study of MLOps within a production ML example.

UNDERSTANDING CI/CD FOR TRADITIONAL SOFTWARE

In traditional software development, CI/CD, which refers to Continuous Integration and Continuous Deployment, is a familiar way of automating workflows for the build, tests, and deploys of code changes. The idea is to verify that code changes can be tested in a non-production environment through an automated pipeline. In a classical context, the software behaviour is deterministic and the outcome is preserved as long as the code remains the same.

A common CI/CD flow looks like this:
Continuous Integration: The developer commits the code to a version control tool (like Git), and the build and unit tests for proper behaviour are triggered.

Continuous Deployment: After the code passes the tests, the application is packaged (often into a Docker container) and deployed automatically to a production environment.

This flow works generally well for standard applications where the unit tests for the output are binary (it will either pass or fail), and also where the external quality of data does not impact the full performance of the application. This assumption of deterministic behaviour does not hold in a CI/CD workflow with a deployment of a Machine Learning model and this presents a limitation for classical CI/CD.

WHY CI/CD IS DIFFICULT FOR MACHINE LEARNING

Incorporating machine learning (ML) into Continuous Integration/Continuous Deployment (CI/CD) adds complexity because ML systems are data and code driven. Traditional CI/CD primarily evaluates the correctness of the source code through unit tests, while the ML model's quality depends upon data quality, feature engineering, hyperparameters, and stochastic training. Essentially, two builds of the same source code may produce different ML model outputs or frame output correctness. Consequently, traditional CI/CD binary pass/fail approaches do not lend themselves to ML deployment and CI/CD practices.

Moreover, machine learning (ML) can be vulnerable to different types of "drift," specifically (1) data drift (change in input data patterns) and (2) concept drift (the change of underlying business rules in the real world). In many situations, a machine learning model that has already been fit to data starts to lose accuracy once new, never-before-seen data is observed. Thus, monitoring and retraining can be considered as a continuous part of the proxy machine learning systems that were designed to assistance in this case and which typical CI/CD pipelines (that apply to software) will not accommodate in the same way. ML pipelines have to deal with large artifacts (datasets, model binaries, experiment metadata, and metrics), and tracking them in version control will require the "data types" used to have all have different systems to support version control (such as DVC, MLflow etc). In addition, resource consumption complicates time and money consumption, as much of training will require GPU clusters to execute training, this would make the time in executing the pipelines easy, as this will be the most expensive part of the pipeline. In short, changes mean ML has the be much more dynamic, metrics, and adaptable with CI/CD integration than the classical CI/CD integration conception.

Thus, integration is more complex, and complicated.

KEY CHALLENGES IN ML INTEGRATION

Incorporating Machine Learning models into CI/CD pipelines creates difficulties that software development generally doesn't think about. These include:

1. Data Versioning:
ML models are very sensitive to the training data; anything prematurely causing a change in the distribution of the data itself can have a drastic impact on the outcome. Thus data needs to be versioned together with the code to ensure the results are reproducible.

2. Model Versioning.
ML artifacts are made up of model weights, feature encoders, and hyperparameter configurations, all storage consuming pieces of software that required versioning and tracking so that we can compare results from experiments.

3. Validating a Metric Instead of a Simple Pass/Fail Test:
You cannot validate ML models using traditional pass & fail test cases. Instead, we should validate model performance with a metric (i.e., accuracy, F1, ROC AUC) and then which model to deploy becomes a thresholding problem.

4. High Compute Cost Needed to Train a Model:
To train and re-train ML models requires GPUs (and often distributed compute) which can increase the amount of time to execute pipelines, not to mention costing extra money to run a pipeline.

5. Model Drift and Concept Drift: Depending on how your model is deployed, performance can degrade for a variety of reasons related to changing patterns in real world data, therefore dynamic retraining will be necessary on an ongoing basis along with monitoring.

6. Complex deployment environments: When serving ML models in production there is a variety of operational complexity to be considered - containerization, scalable infrastructure (e.g. k8s), serving requests in various throughput methods (batch or real-time), etc. This further exemplifies that ML CI/CD has a different and new scale of challenges to consider related to workflows, infrastructure and tools in comparison to traditional software CI/CD.

MLOps AS A SOLUTION

MLOps (Machine Learning Operations) signifies a new engineering culture reflecting the deployment of DevOps practices into the continuous lifecycle of machine learning (ML) systems. MLOps automates the entire ML lifecycle from the insertion of data to the deployment and monitoring of ML models. It combines similar practices as DevOps, alongside new practices of managing experiment tracking, dataset versioning, continuous training, and metrics-driven production gates, for the best-performing model in the experiment to be pushed into production.

Leveraging MLOps, the CI/CD pipeline can be further explored for each of the additional stages (i.e., data validation, feature transformation, training pipeline, model evaluation, and model registry updates) into further CI/CD processes. Tools such as MLflow, DVC, Kubeflow, and Azure ML establish consistent management of these additional stages and/or actions, as well as the reproduction of results across training iterations and experimentation. A robust body of literature shows the MLOps practices support continuous monitoring once deployed, so the ML system can detect data drift or decreasing performance in real-time, and, in turn, self-enact the training workflow to complete retraining of the ML model.
By engineering automation for MLOPs deployment, ML deployment is embedded into an (eventually) automated lifecycle reducing potential human error and is an effective (and future potential) deployment plan for dependable and scalable ML systems in the cloud.

ARCHITECTURAL DESIGN PATTERNS FOR ML CI/CD

MLOps-enabled CI/CD pipelines require an architecture pattern aligned with both software workflows and data workflows. A very common architecture is referred to as three-layer MLOps architecture with the following layers.

1. Data Pipeline Layer: The data pipeline layer is concerned with the ingestion of data, verification of the data, cleaning the data, and extraction of features from the data. The goal is to ensure that any data ingress during this service is of the desired quality. For example, tools such as Great Expectations and Pandera are designed to monitor expectations on incoming data prior to model training.

2. Model Pipeline Layer: The model pipeline layer is for training models, tuning hyperparameters, and tracking model related experiments. The tools used for model management, monitoring and versioning stored model artifacts include, but are not limited to MLflow, DVC, Optuna, or Kubeflow. Each experiment will have metrics tracked to evaluate performance.

3. Deployment Pipeline Layer: The deployment pipeline layer is focused on taking the best performing model artifact, creating a packaged model in a container (such as, Docker), and pursuing that versioned model into a model registry for deployment to a serving platform such as Kubernetes, Seldon, or Azure ML Online Endpoints. Ideally, the model would also have a metrics construct for reject state to ensure that a low-quality model would not be the released model.

This structure enables event-driven automation, where re-training and re-deployment occurs automatically each time new data becomes available or when there is a dip in performance. In this architecture, operational stability, scalability, and reliability of ML operations is across the entire lifecycle of the model.

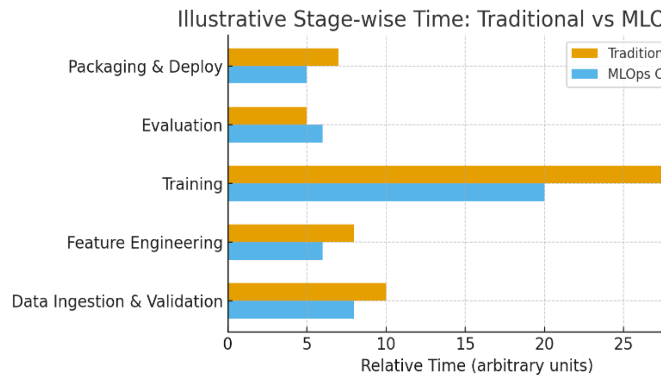

Fig 1 — Stage-wise time comparison between a traditional CI/CD workflow and an MLOps-enabled pipeline.

TECHNOLOGIES & TOOLS SUPPORTING ML CI/CD

A number of specialized tools and systems promote the integral implementation of Machine Learning into CI/CD processes through automated data mobility and management, organizing experiments, and deploying applications.

Versioning Tools:

DVC (Data Version Control) is a tool that can add versioning of datasets and models, similar to the functions of git, while enabling reproducible ML processes while experimenting. MLflow helps track the experiment, manage a model registry, and log parameters all at once, to take advantage of these capabilities for sharing.

Pipeline Orchestration:

Kubeflow Pipelines automate an end-to-end ML workflow for examples of data preprocessing through model training and model evaluation with the functionality for a Directed Acyclic Graph (DAG) based workflow.

CI/CD Platforms:

GitHub Actions, GitLab CI/CD, Jenkins, Azure DevOps can all trigger new builds of models, testing models, and deploying a model's model state when either new events occur in the code or new data arrives.

Containerization & Deployment:

Docker, Kubernetes allow models to be deployed in a portable and scalable manner, while serving frameworks like Seldon Core and Torch Serve allow for the use of models for inference with streaming or batching of incoming data.

In sum, these tools work together to represent some of the strongest ecosystem to manage a fully automated and reproducible ML lifecycle, as a part of broad CI/CD pipelines.

CASE STUDY: ML CI/CD ON AZURE + GITHUB ACTIONS

This research looks at a real-world application of an ML-based CI/CD pipeline built on GitHub Actions, which is connected to Azure Machining Learning and orchestrates the management of model preparation, training and evaluation from the commit/merge of code to a centralized code repository (GitHub).

Pipeline Workflow
Whenever a developer pushes a commit or information about an updated dataset to the GitHub repo, GitHub Actions will automatically perform an Azure ML training job. The Azure ML model will train in the cloud, with the potential to leverage GPU and when complete will log metrics including accuracy, F1 score, and AUC to the MLflow model registry.

Decision Gate
The pipeline implements a metric-based gate for the deployment of the resulting models; if the model meets the performance threshold of the baselines, a new Docker image is created and pushed to the Azure Container Registry and can then be pulled to update the service in Azure Kubernetes Services, if the performance is below the threshold metrics it will not be deployed and alerts will be sent out.
Outcomes & Benefits:
This delivery improved the reliability of machine learning models and made their delivery faster. Manual processes were eliminated and deployment times were reduced by about 60%. Several underperforming models remained out of production. This case study shows that CI/CD driven by MLOps is technically feasible and will have great implications for ML operations in the natural world.

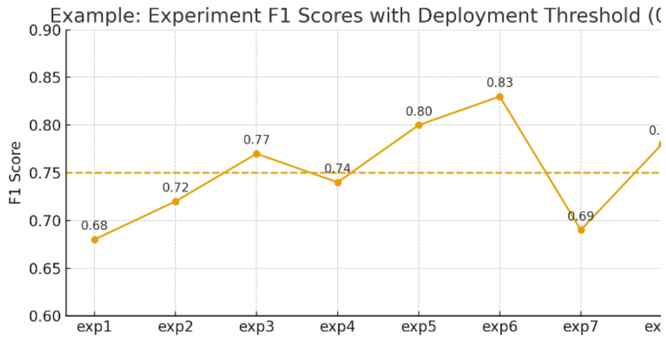

Fig 2 — Example of metric-based deployment gating.

COMPARISON BETWEEN TRADITIONAL CI/CD VS ML CI/CD
The typical CI/CD workflow is focused on the automation of the software delivery workflow, done by detecting changes to the source code, applying automated testing, and deploying compiled artifacts into production. The process is primarily deterministic: if the source code does not change, the observable output behavior will also not change in kind. This suggests that in the CI/CD workflow that, testing is effectively pass/fail and deployment is mostly trivial.
ML CI/CD has a different trajectory because the performance of a model is not only related to the code, but is a function of data and training configurations. Testing is testing an artifact (data sets, model weights, feature encoders, hyper-parameters, log of experiments, and evaluation), so it is not merely a pass/fail test. Testing is fundamentally measuring performance metrics—F1-score, accuracy, AUC, etc. Therefore deploying a model is not merely pass/fail on unit tests but if it meets the evaluation criteria.
To put it succinctly, traditional CI/CD is automating delivery of software code and answering the questions of deployment through the traditional software application framework. ML CI/CD incorporates the entire data-model life cycle with a more robust, pointed and above all, intelligent system that encompasses specialized tools as well as sophisticated versioning strategies, monitoring systems and decision-gates to not only prioritize a model that works to a degree plausibly reasonable, but a performing model.

FUTURE SCOPE IN ML CI/CD
ML-integrated CI/CD is moving towards a future of higher levels of automation, intelligence and cross-cloud portability. One area of development is self-healing pipelines that allow the system to self-detect degradation in models based on signals in the data, automatically initiate re-training, hyper-parameter tuning, or re-sampling datasets, without a human user needing to do anything. Auto rollback features look promising as well, as lower performing a model can be swapped out for a stable model from a prior deployment based on signals from live monitoring. With the distributed workloads being increasingly deployed (Azure, AWS, GCP), cross-cloud portability will be vital as organizations will have predictable deployment across environments. Templates for standard MLOps processes, and model registry formats to be exchanged across different platforms will further enable cross-cloud, cross-environment and cross-deployment portability. Finally, generative

AI and LLM or LLM assist dev tooling to automate a higher level of automated building a pipeline, designing and executing an experiment, implementing evaluation, all based on the degree of assistance and oversight a user may want. The ongoing direction of ML CI/CD will ensure even greater levels of autonomy, optimization, and value, compelling even greater level of cost efficiencies and enabling easy and automated use.

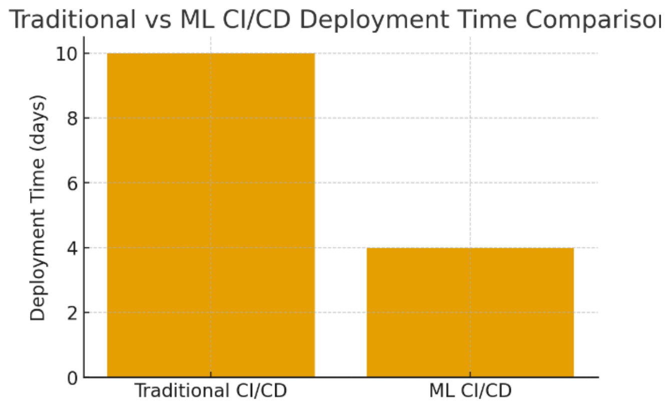

Fig 3-Tranditional vs ML CI/CD DEPLOYMENT TIME COMPARISION

CONCLUSION

The combination of Machine Learning and Continuous Integration/Continuous Deployment heralds a significant departure from traditional DevOps, as ML systems are constantly changing, driven by data, and non-deterministic. We contend that you cannot simply implement CI/CD for ML pipelines due to the challenges of versioning datasets, validating metrics, model drift, and high computational hurdle. In summary, MLOps is a welcome progression of CI/CD that is meant to automate the entire ML lifecycle, or the ML pipeline, which begins with data preparation and extends to model training, validation, deployment and monitoring.

The proposed frameworks and case study illustrate how ML-enabled CI/CD pipelines increase velocity to production, while increasing reliability by preventing deployment of low-quality models. The move to ML will shift MLOps from a 'nice-to-have' to a necessary capability for organisations to maintain accuracy, scalability, and trust in their intelligent solutions.

REFERENCES

[1] D. Sculley et al., “Hidden Technical Debt in Machine Learning Systems,” in *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[2] M. Zaharia et al., “Accelerating the Machine Learning Lifecycle with MLflow,” *Databricks Engineering Blog*, 2018.

[3] D. Crankshaw et al., “InferLine: Latency-Aware Provisioning and Scaling for Prediction Serving Clusters,” in *Proceedings of OSDI*, 2020.

[4] Microsoft, “MLOps v2 Reference Architecture,” *Azure Architecture Center*, Microsoft Docs, 2023.

[5] Kubeflow Project, “KFServing Technical Overview,” *GitHub Documentation*, 2022.

[6] T. Chen, C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” in *Proceedings of KDD*, 2016.

[7] J. Dean et al., “Large Scale Distributed Deep Networks,” in *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[8] T. Breck et al., “The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction,” *Google Research*, 2017.

[9] M. Amershi et al., “Software Engineering for Machine Learning: A Case Study,” in *IEEE/ACM International Conference on Software Engineering (ICSE)*, 2019.

[10] H. F. Alam et al., “A Survey on MLOps: Definition, Taxonomy, Challenges, and Future Directions,” *IEEE Access Journal*, 2023.