MLOps for Research Labs

Vaidheeswaran Archana, Soham Chatterjee

June 2023

0.1 Introduction

Machine Learning Operations (MLOps) is a rapidly evolving field that addresses the need for reliable, efficient, and scalable deployment of machine learning models. It applies principles from DevOps to the unique challenges posed by machine learning, such as data versioning, model tracking, and reproducibility[1]. While MLOps practices have been widely adopted in industry settings, they are less prevalent in research labs, where the primary focus is on model development and experimentation rather than deployment[2]. However, the benefits of MLOps - such as improved reproducibility, streamlined workflows, and enhanced collaboration - are equally relevant to research settings[3]. In this paper, we propose an affordable MLOps architecture specifically tailored for research labs and academia.

Our proposed architecture is built on research labs' fundamental requirements and constraints. These include minimising engineering overhead, maintaining a flow state for researchers, providing cost-effective solutions, ensuring data security, and managing resources efficiently. The architecture also prioritizes critical aspects such as model management and versioning, reproducibility and hyperparameter tracking, data source and versioning, easy iteration and experiment tracking, sharing and collaboration, and balancing automation and complexity.

We provide a detailed explanation of the proposed architecture, which includes components such as a High-Performance Computing (HPC) system, a data store for versioning and tracking data, and a results store for storing model files, training metadata, and artefacts. The architecture also includes provisions for visualizing training results and deploying models as needed.

Finally, we present case studies showing how this architecture can be adapted to different scenarios commonly encountered in research labs. These include scenarios where the lab may not have access to GPU/HPC and wants to fine-tune large language models on a specific task or where a pipeline is needed to deploy the latest model to a particular deployment device automatically. We will also propose an additional scenario of our own, further demonstrating the flexibility and adaptability of the proposed architecture.

The overarching goal of this paper is to demonstrate that with the right architecture and selection of tools, MLOps can be implemented affordably and

effectively in a research setting, leading to improved reproducibility, collaboration, and overall efficiency in the machine learning research process.

1 The Necessity and Adaptation of MLOps in Research Labs

Machine Learning Operations (MLOps) plays a crucial role in enhancing the efficiency, reproducibility, and scalability of machine learning model deployment in industry settings. Yet, its adoption in research labs remains less prevalent despite offering numerous benefits that could significantly streamline processes and improve collaboration. This section of the paper aims to explore why MLOps is needed in research labs and how its priorities can be repurposed to suit the unique requirements of such environments.

1.0.1 Why MLOps is Essential for Research Labs

MLOps addresses the scale and complexity of deploying machine learning models, which is typically necessitated by large businesses catering to vast customer bases. However, research labs primarily focus on model development and experimentation and operate on a smaller scale. The challenge then becomes tailoring MLOps principles to meet the needs of these smaller teams without overwhelming them with processes designed for larger corporations.

In research labs, the primary focus is on improving model performance and innovation rather than deployment. Limited compute resources, which are often a constraint in research labs, necessitate an MLOps framework that optimizes these resources without compromising on the quality and innovativeness of their work.

Furthermore, the engineering overhead in research labs should ideally be minimal, allowing researchers to concentrate more on model development and less on the engineering intricacies of machine learning. A suitably tailored MLOps system should operate largely autonomously without demanding significant maintenance time, thereby reducing disruptions in the researchers' workflow and facilitating uninterrupted experimentation.

Cost is another significant factor in research labs, which often operate with stringent budgets. The MLOps system in such an environment needs to be as cost-effective as possible, leveraging open-source tools to provide powerful research capabilities without incurring substantial expenses.

Lastly, data security and ownership are paramount in research settings, where data might be private, confidential, or sensitive. MLOps systems must ensure robust data security, control access, and safeguard data ownership.

1.1 Prioritizing MLOps Elements for Research Labs

In the context of a research lab, the focus of MLOps shifts from deployment to efficient model management and versioning. This shift enables researchers

to easily track, compare, and analyze their models, thereby improving the lab's overall efficiency. Reproducibility is another crucial priority, requiring thorough tracking of model hyperparameters to ensure consistent and reliable experimental results.

Version control extends beyond models to the data used in experiments. Accurate tracking of data sources and versions is crucial in ensuring that research results are valid and can be reproduced. Tools like DVC can aid researchers in managing their data effectively.

MLOps can enhance the research process by enabling easy iteration and experiment tracking. By incorporating this feature, researchers can keep a detailed record of their experiments, including data, hyperparameters, and results, aiding in result comparison and model improvement.

Sharing and collaboration are other aspects where MLOps can be beneficial. MLOps can stimulate collaboration within and between research labs by facilitating easy sharing of code and research findings. Platforms like GitHub can make it easier for researchers to share their work, fostering a collaborative environment that can drive innovation.

While automation can simplify certain aspects of the research process, an effective MLOps system for research labs should balance automation with simplicity. Researchers should be able to use the system without being burdened by excessive complexity that could deter their research efforts.

The proposed MLOps architecture outlined in this paper seeks to cater to these unique requirements and constraints of research labs, offering a cost-effective, efficient, and collaborative environment for machine learning research. By understanding and adhering to the workflow of a typical research project, this architecture ensures that MLOps can be effectively adapted to the needs of research labs, enabling them to leverage the many benefits of MLOps without being overwhelmed by its complexity.

2 Proposed MLOps Architecture for Research Labs

Given research labs' unique requirements and constraints, we propose an MLOps architecture tailored specifically for such settings. This architecture is built around the typical research project workflow, which includes data gathering, analysis, model creation and training, results visualization and analysis, iteration, result sharing, and potential model deployment.

- Data Gathering and Analysis: The first step in the research project workflow involves acquiring and studying the data. The raw data undergoes a process of cleaning and preprocessing for subsequent model training. It's essential to version this data and record the preprocessing steps for reproducibility. As the project evolves, researchers might apply different preprocessing techniques, creating alternative data versions, all of which should be carefully documented.

- Model Creation and Training: Once the data is ready, researchers proceed to construct and train the model architecture. Recording the model hyperparameters, model files, and key metrics throughout the process is crucial. Additionally, each training run should be associated with the data version used.
- Model Creation and Training: Once the data is ready, researchers proceed to construct and train the model architecture. Recording the model hyperparameters, model files, and key metrics throughout the process is crucial. Additionally, each training run should be associated with the data version used.
- Results Visualization and Analysis: Researchers need to visualize and examine the results after training. If the outcomes are not up to expectations, they may iterate the process, either modifying the model or further processing the data.
- Iteration: Iterative cycles form an integral part of the research process. Every step in these cycles - from data preparation, model training, and parameter tuning, to results analysis - should be tracked meticulously to aid further improvements and ensure reproducibility.
- Result Sharing: Whether sharing preliminary or final results, tracing all the steps leading to these outcomes is vital. This includes the data version, preprocessing steps, model files, training parameters, and all analyses and results. Even failed attempts and experiments that guided the researchers to the final model should be readily accessible for sharing.
- Model Deployment: Although not always necessary in a research setting, there might be instances where researchers want to deploy their model as a demonstration. This process should involve retrieving the model files from the model store and deploying them to an appropriate platform.

The proposed MLOps architecture includes several key components:

- High-Performance Computing (HPC): Located centrally, the HPC is where researchers submit their training jobs. The HPC should be capable of fetching data from the data store.
- Data Store: Here, researchers can clean, process, and store data. Any data used for a training run should be versioned and tagged with the corresponding training run.
- Result Store: After training, the model files, along with all training metadata and artifacts, should be stored in the result store. Researchers can then access this store to analyze their results.
- Front-End (Optional): A front-end can be included to visualise training results. If the model is deployed, the front end can also display results from the model.

Based on this architecture, we suggest a sample implementation using popular tools.

DVC is employed for versioning data. Perfect is used for orchestrating tasks running on the HPC. Jobs can be submitted to the HPC using GitHub Actions. Training jobs can write their result to MLFlow, which tracks the experiment metadata and serves as a model registry. MLFlow can also deploy models to SageMaker or other deployment platforms. Plotly can be used to display the results of model training. It can also query models in SageMaker to display results. In summary, the proposed MLOps architecture seeks to provide research labs with an efficient, cost-effective, and easy-to-implement solution that aligns with their unique workflow and requirements. This architecture should help research labs streamline their processes, ensure reproducibility, and facilitate seamless collaboration and sharing of research outcomes.

3 Detailed Analysis of Tools for MLOps in Research Labs

In this section, we delve into an in-depth examination of the various tools suited to the MLOps needs of research labs. We evaluate each tool in the context of its intended function within the MLOps workflow, discussing its unique advantages and potential limitations.

3.1 Data Labelling: LabelStudio and CVAT

Data labelling is an essential initial step in most machine learning workflows, where raw data is annotated to provide ground truth for the training of models.

LabelStudio provides diverse functionalities that support various labelling tasks, making it adaptable to various data types. Its flexibility, coupled with its extensibility due to the open-source nature, makes it a powerful tool for different labelling needs. Moreover, LabelStudio offers a customizable and user-friendly interface, enhancing usability. However, its open-source nature means users might need to dedicate substantial time to set up, customize, and maintain the software. Furthermore, for very specific labelling tasks, such as detailed image annotation, it may not be as efficient as specialized tools like CVAT.

CVAT shines when it comes to the detailed and efficient annotation of image data for computer vision tasks. Its user interface is designed to expedite the annotation process, thereby increasing productivity. Moreover, its open-source nature allows for customization to suit specific needs. On the downside, CVAT's limited applicability to non-vision tasks may pose restrictions, and setting it up might require a certain level of technical expertise.

3.2 Data Versioning: DVC

DVC stands out for its ability to handle large datasets and model files. Its integration with Git creates a familiar workflow for developers and allows the

versioning of data and models alongside code. DVC supports a wide range of storage options, adding flexibility. However, its command-line interface might be difficult for non-technical users to navigate. Additionally, while DVC does a great job in data and model versioning, it might not suffice for more extensive MLOps needs, requiring additional tools to form a complete MLOps pipeline.

3.3 Feature Store: Feast

Feast's key advantages lie in its ability to provide consistent feature values across different ML stages and environments, reduce data duplication, and enable back-testing with historical feature values. It also supports real-time feature serving, making it useful for online ML models. On the other hand, Feast may pose a challenge for smaller teams as its setup and management require substantial effort and understanding of concepts like feature engineering and data infrastructure.

3.4 Model and Experiment Versioning and Tracking: MLFlow, DVC

MLFlow offers comprehensive tools for managing the end-to-end machine learning lifecycle. Its experiment tracking feature allows efficient logging and comparison of different runs, aiding reproducibility and collaboration. On the other hand, MLFlow might be an overkill for smaller projects, as setting up and maintaining it can be complex. It also might not fully support non-Python workflows.

While DVC primarily excels in data versioning, it also offers model versioning and experiment tracking functionalities. It provides a lightweight solution for smaller projects or for teams primarily interested in versioning. However, for a more detailed and broader scope of tracking, tools like MLFlow might be more suitable.

3.5 Code Repository and Pipelines: GitHub and GitHub Actions

GitHub is the de facto standard for code hosting, with a robust ecosystem and integrations, making collaboration easier. However, it may not offer the level of control or customization that some projects may need. Some teams may also have privacy concerns, especially with public repositories.

GitHub Actions provides automation capabilities directly within the GitHub environment, which is a major advantage for teams already using GitHub. It supports many tasks, from simple CI/CD workflows to more complex ones. However, for intricate workflows or specific CI/CD requirements, dedicated CI/CD solutions might offer more functionalities.

3.6 Orchestration: Prefect, Metaflow

Prefect provides a user-friendly Pythonic interface for building complex workflows. Its cloud-based UI allows for easy monitoring of tasks. However, running Prefect at scale may require dedicated infrastructure and management.

Metaflow simplifies many aspects of managing data science projects, such as dependency management, versioning, and execution of workflows. Its human-centric design and integration with AWS services make it a powerful tool for data scientists. However, it is mainly optimized for AWS, which might limit its usage in non-AWS environments.

4 Implementation

Our proposed MLOps architecture's implementation stage forms the backbone of the infrastructure, enabling seamless integration of different components to facilitate the machine learning lifecycle. As this phase's demands vary significantly across projects, we shall describe a generalized approach to help guide the implementation, which can be customized as per project specifics.

The first step entails setting up data labelling tools, such as LabelStudio or CVAT, depending on the type of data that needs annotation. These tools should be configured to meet the precise annotation requirements for different types of data, like images, text, or audio. For LabelStudio, users need to define templates for specific labelling tasks, which can range from simple classification to more complex structures like bounding boxes or polygonal segmentations for images.

Next, the setup and integration of data versioning tools like DVC come into play. Here, we must establish a remote storage location, such as an AWS S3 bucket or a GCP Storage bucket, and link it with the local DVC setup. This remote storage will host the data and model files, thereby preserving different versions of these assets. Integration with Git allows us to version control our data alongside code, providing a comprehensive view of the project's evolution.

Setting up the feature store, such as Feast, involves the installation and configuration of the tool and the definition of features. Here, features should be carefully engineered and stored in the Feast store to provide consistent feature values across different ML stages and environments.

For experiment tracking and model versioning, MLFlow and DVC can be used. While DVC excels in versioning data and models, MLFlow allows efficient logging and comparison of different runs. The setup of these tools involves configuring the experiment tracking server for MLFlow and setting up a repository for DVC to manage model versioning. It is essential to define a clear naming convention and directory structure for efficient tracking.

In terms of the code repository and pipelines, we propose GitHub and GitHub Actions. The setup of GitHub involves creating a repository and defining the structure and workflow. Setting up GitHub Actions involves creating specific YAML files in the .github/workflows directory, defining the CI/CD

pipeline stages, and setting up triggers for each stage.

Finally, orchestration tools like Prefect or Metaflow are used to manage and automate the ML workflow. These tools require a detailed definition of the tasks, dependencies, and triggers.

5 Case Studies

We present the following case studies to demonstrate how our proposed MLOps architecture can be moulded to fit different scenarios. These represent common situations in research labs and illustrate how the architecture can be flexibly adapted to meet different needs.

5.1 Case Study 1: Collaborative Research with Multiple Researchers

Many research labs engage multiple researchers working on different aspects of a project. This collaboration could be hampered by lack of proper tools or mechanisms to share progress, leading to overlapping efforts or miscommunication. For example, two researchers might independently tune the same model parameters, wasting precious compute resources and time.

A cooperative MLOps setup could look like this:

- Collaborative Code Development: GitHub serves as the primary platform for sharing and versioning the code. All researchers can contribute to the codebase, and any changes are clearly documented through Git's versioning system. This also facilitates code reviews and discussions.
- Data and Model Versioning: DVC and MLFlow ensure that all researchers are working with the same data versions and can track each other's progress in model development. Any new data preprocessing or feature engineering techniques can be shared through DVC, while MLFlow allows researchers to track each other's progress on model training.
- Result Sharing and Collaboration: A shared dashboard on Plotly can visualize the model performance metrics logged in MLFlow. This allows all researchers to easily monitor the project's progress, facilitating discussions and collective decision-making.

6 Case Study 2: Fine-tuning Language Models without GPU/HPC

In this scenario, a research lab is tasked with fine-tuning a large language model (LLM) for a specific NLP task, but they lack access to a high-end GPU or High-Performance Computing (HPC) system. Without these powerful computing resources, training or fine-tuning an LLM could take a prohibitive amount of

time. This predicament is not uncommon in smaller labs or labs in developing countries where budget constraints restrict access to high-end hardware.

To overcome these challenges, this is how the MLOps architecture could be implemented:

- Cloud-Based Training: The lab can use cloud computing platforms like AWS, Azure, or Google Cloud Platform (GCP), which provide virtual machines with high-end GPUs that are ideal for training machine learning models. In this setup, the researchers would create a training script for fine-tuning the LLM and use Prefect to manage the execution of this script on the cloud platform. The training script can be version-controlled using GitHub.

- Data and Model Versioning: Despite the computational constraints, maintaining the reproducibility of the work is crucial. DVC can handle the versioning of the training and validation datasets, ensuring that the exact data splits used for each experiment are recorded. MLFlow is used for logging the model parameters, hyperparameters, and the model performance metrics.

- Result Analysis and Deployment: After training, the results can be pulled from MLFlow and analyzed. Plotly can serve as a powerful tool for visualizing the model's performance and comparing it across different runs. Once an optimal model is found, it can be deployed to a service like AWS SageMaker, where it can be accessed via API calls.

7 Case Study 3: Automated Pipeline for Model Deployment

In this case, a research lab is working on a project requiring frequent model updates. For example, they could develop an anomaly detection system that needs to be regularly retrained with new data. Manually redeploying the model every time it's updated can be a tedious and error-prone process, increasing the likelihood of introducing bugs or deployment issues.

The following architecture can be adapted to automate the deployment:

- Automated Model Training and Deployment: GitHub Actions can be used to monitor changes to the model training script. Whenever a change is detected, it can trigger a training job on the HPC or a cloud platform, depending on the resources available. After training, the model and its metadata are logged to MLFlow.

- Continuous Deployment: Upon successfully completing a training job, another GitHub Action can be triggered to deploy the updated model to the deployment platform. Depending on the use case, the deployment platform could be AWS SageMaker, Google AI Platform, or an edge device like

a Raspberry Pi. MLFlow can manage this deployment process, ensuring that the correct model version is deployed each time.

- Monitoring and Alerts: Prefect can be used to monitor the automated pipeline's health. This tool provides visual cues about the status of different tasks and can send alerts if tasks fail or underperform.

In all these case studies, the underlying principle is to adapt the MLOps architecture to facilitate research activities rather than hinder them. The flexibility and modularity of the proposed architecture allow it to cater to diverse needs, enhancing the efficiency and reproducibility of research.

8 Alternate Tools in Building MLOps Systems

While we discussed a particular set of tools in the context of our proposed MLOps architecture, several other alternatives are also popular, potentially offering more cost-effective or suited options.

With its active learning approach, data labelling tools like Prodigy provide an efficient and budget-friendly alternative. For Data Versioning, Pachyderm is an excellent option that provides a version-controlled data lineage system and works well for data scientists and engineers. For feature stores, Tecton provides a robust and enterprise-ready solution.

TensorBoard is an excellent tool for experiment tracking that visualises machine learning experiments. It can be combined with TensorFlow, PyTorch, or any other deep-learning framework. Neptune.ai, another notable alternative, offers experiment tracking and model registry services in a user-friendly interface and is highly extensible.

Regarding CI/CD, Jenkins provides a powerful, flexible, and open-source automation server that allows efficient building, testing, and deploying applications. GitLab CI/CD is another tool that integrates seamlessly with the GitLab ecosystem and can be an excellent choice if the team is already using GitLab for version control.

For orchestration, Apache Airflow provides a programmable scheduling and monitoring tool that's especially useful in data engineering tasks. Argo, which works within Kubernetes, provides a solution for complex workflows, particularly when a project requires a significant degree of container orchestration.

9 Conclusion

Integrating MLOps practices within research labs can significantly improve machine learning projects' efficiency, scalability, and reproducibility. This paper proposed an MLOps architecture designed to cater to a research lab setting and explored the various tools required for effective implementation.

The architecture we proposed addresses key aspects of MLOps, such as data labelling, data and model versioning, feature stores, experiment tracking, code

repositories, CI/CD pipelines, and orchestration. We have tried to showcase the versatility of this design by highlighting its ability to accommodate various tools, thus making it adaptable to different scenarios and needs.

We believe that this architecture is well-suited to aid research labs in managing their machine-learning projects effectively. However, with the evolving nature of machine learning research and the burgeoning growth of MLOps tools, it is crucial to keep exploring and experimenting with new strategies and tools. Our proposal provides a guiding architecture to set up MLOps systems. Still, the constant endeavour for innovation will lead to the most efficient systems tailored to individual project needs.

In conclusion, we believe that this MLOps architecture offers an effective path towards enhancing the efficiency, transparency, and reproducibility of machine learning projects in research labs. The architecture's underlying principle is to ensure that machine learning development is not an isolated process but is tightly integrated with the entire software development lifecycle, promoting collaborative efforts and scientific discoveries.