Experiment Tracking and Optimisation Using Mlops

Harish Kumar K S[1], Y Amarnath Chowdhary[2], Charan G[3], Deekshith K A[4], K R Vishnu Kumar[5,] Vignesh R [6]

[1] *Associate Professor, Dept. of Computer Science and Engineering, Presidency University, Bengaluru, Karnataka, India*

[2,3,4,5,6]*UG Student, Dept. of Computer Science and Technology, Presidency University, Bengaluru, Karnataka, India*

***Abstract*—We suggest the creation and deployment of an experiment tracking and optimization system based on MLOps principles. The system will utilize DVC (Data Version Control) to effectively manage and track a variety of machine learning experiments by changing hyperparameters, configurations, and workflows. In contrast to conventional model-centric methods, our emphasis is on the experimentation process itself, guaranteeing reproducibility, scalability, and automation in ML workflows. The system to be proposed will support effortless versioning of experiments, automatic tracking of results, and organized optimization of hyperparameters so that comparing different runs becomes less complicated and identifying the optimal configuration becomes more feasible. With Git for versioning and DVC for pipeline control, we set up a sound framework for dealing with iterative experimentation. This work describes the design, deployment, and assessment of our MLOps-driven experiment tracking system, emphasizing its advantages in enhancing workflow reproducibility, performance tracking, and parameter tuning. Furthermore, we discuss possible extensions including automatic hyperparameter tuning, richer visualization of experiment output, and incorporation with cloud-based ML systems. Our methodology is designed to simplify the ML development process, making the process of experimentation more efficient, transparent, and scalable.**

***Index Terms*—MLOPs, DVC, ML**

1. INTRODUCTION

In ML development, the capability to effectively track, handle, and optimize experiments is important in order to realize high-performance models. Traditional ML workflows are normally unsophisticated with regard to structured experiment tracking, resulting in complexities in reproducibility, parameter adjustment, and comparison of performances. To meet such challenges, MLOps as a methodology involving the adoption of DevOps principles in ML workflows has emerged to ensure automation, scalability, and versioning.

This project centers on experiment tracking and optimization with MLOps principles, particularly utilizing Data Version Control (DVC). In contrast to conventional model-centric methodologies that focus mainly on dataset preparation and model training, our system is built to track and optimize ML experiments by controlling hyperparameter variations, configuration changes, and workflow changes. This makes each experiment iteration reproducible, enabling data scientists and ML engineers to make data-driven decisions based on organized insights.

DVC is a central component in our suggested system as it allows version control of experiments, automatic tracking of metrics, and effective pipeline management. By combining DVC with Git, we create a solid foundation that guarantees each experiment's configuration, code, and results are accurately logged and can be easily reproduced. We also investigate the integration of hyperparameter tuning methods, allowing automatic optimization for better model performance.

This paper describes the design, implementation, and assessment of an MLOps-driven experiment tracking and optimization platform. We provide an overview of the limitations in conventional ML experiment tracking, the benefits of our solution, and propose future enhancements, including advanced visualization capabilities, cloud-based integration, and automatic hyperparameter tuning frameworks.

The system in question seeks to automate ML experimentation, improving workflow efficiency, reproducibility, and performance monitoring,

ultimately to result in more organized and stable ML model creation.

2. LITERATURE SURVEY

Experiment Tracking and MLOps

- MLOps streamlines the lifecycle of machine learning models, from development to deployment.
- DVC, MLflow, and Kubeflow are used most frequently for experiment tracking.
- Research emphasizes the importance of reproducibility, scalability, and automation in ML pipelines.

Data Version Control (DVC) Experiment Tracking

- DVC allows data, code, and experiment tracking in a Git-like way.
- Evidence indicates that DVC enhances cooperation and reproducibility in ML projects.
- Combining DVC with hyperparameter tuning results in effective experiment management.

Hyperparameter Optimization in MLOps

- Conventional tuning techniques such as grid search are computationally costly.
- Bayesian optimization and evolutionary algorithms enhance efficiency.
- Research indicates that auto hyperparameter tuning pipelines improve model performance.

Comparative Analysis of Experiment Tracking Tools

- DVC vs. MLflow vs. Weights & Biases: Scalability vs. automation vs. cloud integration trade-offs.
- Research emphasizes DVC's adaptability in fitting into current ML pipelines.
- Experiment tracking tools must balance usability, cost, and deployment feasibility.

Challenges and Future Trends in MLOps Experimentation

- Key issues: Scalability, metadata management, and security.
- Upcoming trends involve AI-optimized experiments and decentralized ML pipelines.
- New studies investigate serverless MLOps and real-time experiment tracking.

3. METHODOLOGY

➢ Proposed Methodology:

The system under proposal targets experiment tracking and optimization based on MLOps principles with DVC (Data Version Control). Our method is designed such that versioning, tracking, and comparison of various machine learning experiments by changing hyperparameters, configurations, and workflows are carried out in a hassle-free manner.

The methodology follows these key steps:

1. Environment Setup & Version Control:
- Install and set up DVC and Git for versioning experiments.
- Establish a formal workflow for ML experiment tracking.

2. Experiment Configuration & Parameterization:
- Create a parameterizable pipeline where hyperparameters and model settings can be varied.
- Define experiment parameters dynamically through YAML or JSON files.

3. Experiment Tracking with DVC:
- Use DVC's experiment tracking to track various runs.
- Make it reproducible by saving all the experiment metadata.

4. Hyperparameter Tuning:
- Make parameter tuning automatic through grid search, random search, or Bayesian optimization.
- Compare and determine the best-performing configurations.

5. Performance Logging & Evaluation:
- Log experiment outcomes, such as metrics, model results, and configurations.
- Analyze performance trends using visualization tools.

6. Versioning & Reproducibility:
- Save all variations of experiments through DVC versioning.
- Facilitate simple rollback to earlier configurations and settings.

7. Automation & Scalability:
- Incorporate automation scripts for simplifying experiment runs.
- Provide scalability for bigger datasets and cloud-based running.

➢ SYSTEM DESIGN:

The system design is made up of several modules that collaborate in order to provide effective experiment

tracking and optimization. The architecture has a modular style, which gives it flexibility as well as scalability.

System Architecture Overview:

➢ Version Control Layer:

- Git + DVC to manage and track experiment versions.
- Stores dataset versions, experiment metadata, and results.

➢ Experiment Management Layer:

- Configuration files to define parameters and experiment settings.
- Scripts for automating different experiment runs.

➢ Tracking & Logging Layer:

- DVC experiment logs to store performance metrics.
- Comparison tools for evaluating different experiment outcomes.

➢ Optimization Layer:

- Hyperparameter tuning framework (Optuna/Hyperopt/Grid Search).
- Automated tuning and performance benchmarking.

➢ Visualization & Monitoring Layer:

- Tools for tracking performance trends.
- Dashboards for real-time experiment monitoring.

Architecture

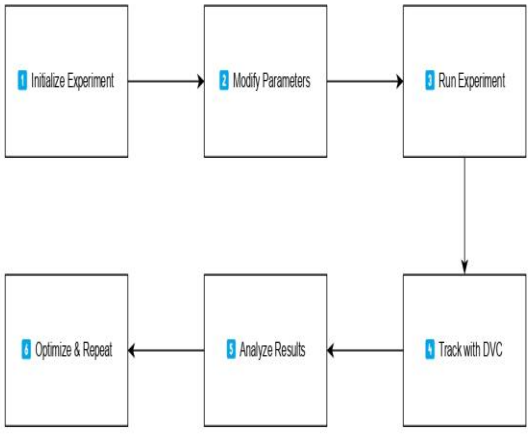

Figure 3.1: Workflow

1. Initialize Experiment

The first step is to set up the environment for running and tracking machine learning experiments. This includes:

- Setting Up DVC and Git:
- Initialize a Git repository (git init) to version control the code.
- Set up DVC (dvc init) to track experiments, data, and configurations.
- Create a .dvcignore file to exclude unnecessary files from tracking.
- Defining Experiment Structure:
- Create a config.yaml or params.json file to store hyperparameters and configurations.
- Define the ML pipeline (preprocessing, training, evaluation).
- Ensure reproducibility by fixing dependencies in requirements.txt or using virtual environments.
- At the end of this step, the environment is ready to run multiple experiments while maintaining version control.

2. Modify Parameters

Once the baseline setup is complete, we modify hyperparameters and configurations for experimentation. This step includes:

- Manual Parameter Adjustment:
- Change hyperparameters in config.yaml or params.json.
- Example: Adjust learning rate, batch size, optimizer type, dropout rate, etc.
- Automated Hyperparameter Search:
- Use tools like Optuna, Hyperopt, Grid Search, or Random Search to explore different hyperparameter combinations systematically.
- Example: Running a grid search over learning rates [0.001, 0.01, 0.1] and batch sizes [32, 64, 128].
- Tracking Configurations:
- DVC ensures every experiment has a unique identifier, making it easy to revisit previous configurations.
- Store each configuration as a separate branch or commit in Git.

By the end of this step, we have multiple experiment variations ready for execution.

3. Run Experiment

This step involves executing the machine learning experiment based on the modified parameters. Key tasks include:

- Running the Training Pipeline:

- Execute the script that trains the model using the updated parameters (python train.py --config config.yaml).
- Store the output (metrics, logs, model artifacts).
- Using DVC Pipelines (Optional but recommended):

Define a reproducible pipeline using dvc.yaml (e.g., data processing → training → evaluation).

Example:

```
stages:
  preprocess:
    cmd: python preprocess.py
    deps:
      - raw_data.csv
      - preprocess.py
    outs:
      - processed_data.csv
  train:
    cmd: python train.py --config config.yaml
    deps:
      - processed_data.csv
      - train.py
    outs:
      - model.pkl
      - metrics.json
```

This structure ensures that if any dependency changes, only necessary parts of the pipeline are re-executed.

At the end of this step, the experiment has been executed and produced results.

4. Track with DVC

Once the experiment is completed, we use DVC to track results and version control changes.

- Saving Experiment Outputs:
- Store metrics (accuracy, loss, precision, recall) in a JSON file (metrics.json).
- Save model artifacts (model.pkl, weights.h5).
- Track logs for debugging (log.txt).
- Versioning Experiments with DVC:
- Use dvc add to version experiment results.

Example:

- dvc add metrics.json model.pkl
- git commit -m "Experiment with LR=0.01, batch=64"
- Push results to remote storage (dvc push).
- Comparing Experiments:
- Use dvc metrics diff to compare performance across different experiments.

Example output:

- Path Metric Old Value New Value Change
- metrics.json accuracy 0.85 0.89 +4.7%

This step ensures that all experiments are logged, versioned, and reproducible.

5. Analyze Results

After running multiple experiments, we analyze and compare results to identify the best-performing configuration.

- Experiment Comparison Using DVC:
- Use dvc exp show to view all experiments.

Example output:

Experiment 1:

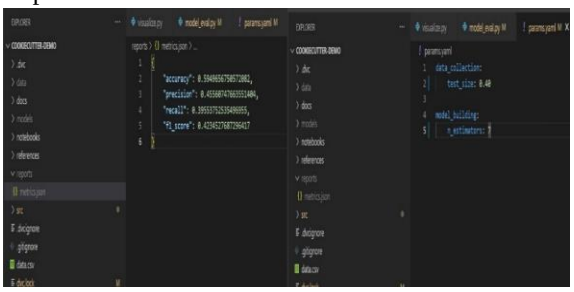

Figure 3.2: For test_size of 0.40 & n_estimators given 7

Experiment 2:

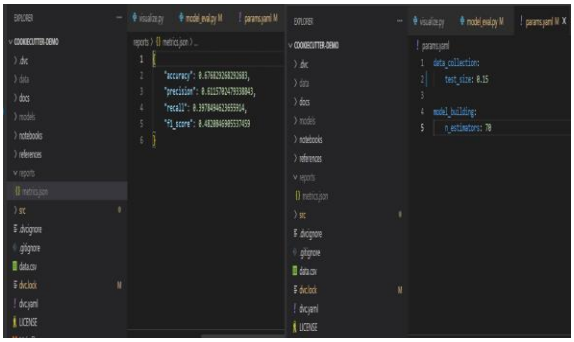

Figure 3.3: For test_size of 0.15 & n_estimators given 70

- Visualization:
  - Use Matplotlib/Seaborn/TensorBoard to visualize accuracy and loss curves.
  - Example: Plot training loss vs. epochs to detect overfitting.
- Root Cause Analysis (if needed):
  - If a model underperforms, analyze training logs and hyperparameters.

- Check data preprocessing, learning rate schedules, and optimizer choices.

At the end of this step, we have identified the best-performing configuration and can proceed to optimization.

6. Optimize & Repeat

Once we analyze results, we perform optimizations and repeat the cycle.

- Refining Hyperparameters:
- If the best experiment had LR=0.001, try finer adjustments (0.005, 0.0005).
- Use Bayesian Optimization or Genetic Algorithms for automated tuning.
- Improving Model Performance:
- If accuracy is low, try data augmentation, feature selection, or ensembling.
- If training is unstable, adjust batch normalization, dropout rates, or learning rate schedules.
- Automating the Process:
- Integrate CI/CD pipelines (e.g., GitHub Actions) to trigger experiments automatically when changes occur.
- Example: When a new dataset is added, trigger a DVC pipeline execution automatically.
- Finalizing the Best Experiment:
- Once an optimal configuration is found, lock the best model version using dvc freeze.
- Store final experiment results in a centralized registry (e.g., MLflow, Weights & Biases) for further use.

4. USE CASE

Experiment tracking and optimization are critical components of the machine learning lifecycle, especially in model development and hyperparameter tuning. The primary challenge faced by data scientists and ML engineers is the lack of efficient experiment tracking, version control, and performance comparison across multiple iterations. This use case focuses on how DVC (Data Version Control) streamlines this process in an MLOps pipeline.

Use Case 1:

Hyperparameter Optimization for a Deep Learning Model

Scenario:

A data scientist is developing a deep learning model for image classification. To improve model performance, multiple experiments must be conducted by varying:

- Learning rate
- Batch size
- Number of layers
- Activation functions

Challenges Without Experiment Tracking:

1. Inconsistent Results – No structured way to track which hyperparameter combination performed best.
2. Reproducibility Issues– Difficulty in replicating successful experiments due to lack of version control.
3. Manual Logging Errors– Tracking performance metrics manually is error-prone and inefficient.

Solution Using DVC:

1. Initialize Experiment Tracking – Set up a structured pipeline using dvc init and define experiment parameters in config.yaml.
2. Modify Hyperparameters – Adjust parameters dynamically and maintain multiple experiment versions.
3. Run Experiments – Train models with different configurations and save results in metrics.json.
4. Track & Compare – Use dvc exp show to compare accuracy, loss, and other key metrics.
5. Optimize & Deploy – Select the best-performing model and integrate it into production.

Outcome:

- Automated hyperparameter tuning leads to improved model performance.
- Reproducibility ensures experiments can be rerun effortlessly.
- Efficient comparison of different configurations saves time and effort.

Use Case 2:

Collaborative Model Development in a Team

Scenario:

A team of ML engineers is working on a fraud detection model. Each member runs different experiments to fine-tune the model. The challenge is tracking individual contributions, avoiding conflicts,

and ensuring reproducibility.

Challenges Without DVC:

1. Experiment Conflicts – Team members may override each other's changes.
2. Loss of Best Results – No structured way to store or retrieve optimal configurations.
3. Difficulty in Merging Experiments – Manually combining results is time-consuming

.

Solution Using DVC:

1. Each Engineer Works on a Separate Experiment – DVC allows team members to run isolated experiments.
2. Tracking Changes Using Git & DVC – Every experiment is version-controlled and pushed to a shared repository.
3. Comparing Team Results – Use dvc metrics diff to identify the best experiment.
4. Finalizing the Model – The best-performing model is locked and moved to production.

Outcome:

- Efficient collaboration among team members.
- Seamless merging of experiments with proper versioning.
- Improved model performance through systematic experimentation.

Use Case 3:

Continuous Model Optimization in Production

Scenario:

A company is using an ML model for real-time sentiment analysis. Model performance degrades over time due to changes in user behavior and data distribution. Continuous monitoring and optimization are required.

Challenges Without MLOps:

1. Model Drift – No tracking mechanism to detect performance degradation.
2. No Version Control – Difficult to revert to a previously better-performing model.
3. Lack of Automation – Manual updates lead to downtime and errors.

Solution Using DVC:

1. Monitor Model Performance – DVC stores performance metrics for each version.
2. Automatically Retrain with New Data – The pipeline is triggered when new data arrives.
3. Compare New vs. Old Models – Use dvc exp diff to check if the new model is better.
4. Deploy Only the Best Model – If performance improves, the updated model replaces the old one.

Outcome:

- Proactive model updates based on real-world data.
- Automated experiment tracking ensures minimal manual intervention.
- Downtime is reduced, improving user experience.

5. CONCLUSION:

This paper introduces the presentation of two new medical question and answer systems, In this project, we implemented an MLOps pipeline for experiment tracking and optimization using DVC (Data Version Control) to streamline the management of machine learning experiments. By focusing on parameter tuning, performance evaluation, and reproducibility, we established a structured six-step workflow: Initialize Experiment → Modify Parameters → Run Experiment → Track with DVC → Analyze Results → Optimize & Repeat. This approach eliminates manual tracking errors, ensures reproducibility, and enables seamless collaboration among team members. With DVC and Git integration, experiments are version-controlled, making it easy to compare and merge results efficiently. The pipeline also supports continuous optimization, allowing iterative improvements by analyzing past performance and selecting the best-performing configurations. Additionally, by automating model tracking and monitoring, our approach enhances scalability in production, reducing downtime and improving ML model performance over time.

This MLOps framework significantly improves the machine learning development lifecycle by reducing experimentation time, optimizing model performance, and ensuring transparency. As a next step, integrating hyperparameter tuning frameworks like Optuna or Ray Tune and CI/CD pipelines could further enhance automation. Ultimately, leveraging DVC-powered MLOps provides a scalable, reliable, and automated

solution for managing machine learning experiments effectively.

REFERENCES

[1] Zhao, H., & Liu, Y. (2020). "Machine Learning Experiment Management: A Survey." 2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA). This paper surveys various tools and methodologies for managing machine learning experiments, including version control systems like DVC.

[2] Studer, S., Bui, T. B., Drescher, C., Hanuschkin, A., Winkler, L., Peters, S., & Mueller, K. R. (2020). "Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology." arXiv preprint arXiv:2003. 05155.The authors propose a process model for machine learning with integrated quality assurance, discussing tools like DVC for data and model versioning.

[3] Schelter, S., Böse, J. H., Kirschnick, J., Klein, T., & Seufert, S. (2018). "Automating Large-Scale Data Quality Verification." Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.This paper addresses the automation of data quality verification in machine learning pipelines, highlighting the importance of version control systems such as DVC.

[4] Matsubara, Y., & Sakurai, Y. (2020). "MLOps: A New Organizational Model for Effective Data Science Collaboration." 2020 IEEE International Conference on Big Data (Big Data). The authors introduce MLOps as an organizational model to enhance collaboration in data science teams, emphasizing the role of tools like DVC in experiment tracking.

[5] Hamzah, H., & Sulaiman, M. N. (2021). "Data Version Control for Reproducible Machine Learning Experiments." 2021 IEEE 11th International Conference on System Engineering and Technology (ICSET). This paper discusses the implementation of data version control using DVC to ensure reproducibility in machine learning experiments.