Designing Reliable ML Pipelines: A Practical Approach to Reproducibility Using Git, DVC, Docker, and CI/CD

Title: *Experimentation and Reproducibility in AI: A Pipeline-Based Implementation Approach*

Submitted by: Manika Singh, Seema Verma, Sabha Ambrin, Gaurav Singh Pariha

---

1. Introduction

In machine learning, building an accurate model is only part of the process — ensuring that others can understand, rerun, and trust that process is what truly makes the work valuable. Reproducibility is the practice that turns a one-time result into a reliable, repeatable workflow. It enables machine learning experiments to be verified, reused, and built upon — not just by the original team, but by any future collaborator or researcher.

This project was designed with one core goal: to develop a machine learning pipeline that could be reproduced across systems, by different people, at different times — with the same outcomes. Instead of treating reproducibility as an afterthought, we placed it at the heart of our workflow by incorporating tools like Git, DVC, Docker, and GitHub Actions from the start. Each tool played a specific role in versioning code and data, controlling the environment, and automating the validation of our experiments.

In the sections that follow, we outline the tools, decisions, and strategies we used to build a complete, reproducible ML project, along with the challenges we encountered and the lessons we learned as a team.

2. Why Reproducibility Matters in Machine Learning

Reproducibility is not just a technical concern; it is a foundational principle for building trustworthy, verifiable, and scalable machine learning systems. Machine learning models often depend on data quality, preprocessing steps, model architecture, random initialization, and even hardware or environment settings. Without proper control and documentation, repeating the same results becomes nearly impossible — even for the original developers.

In both academic and industrial settings, reproducibility plays a central role:

- **In research, it supports peer validation, academic integrity, and the ability to extend prior work.**

- **In industry, it ensures consistent model behavior across development, testing, and deployment environments.**

- **In regulated domains like healthcare or finance, reproducibility is necessary for compliance, audits, and user trust.**

Beyond reliability, reproducibility also improves collaboration. It helps teams understand one another's work, identify bugs faster, and build on shared experiments without conflict.

For these reasons, reproducibility in machine learning is essential — not optional.

3. Tools and Technologies Used

To implement and ensure reproducibility in our machine learning workflow, we integrated a set of modern tools, each addressing a specific aspect of versioning, automation, and consistency.

- **Git and GitHub** were used to manage the source code, track changes over time, and collaborate effectively across the team. Version control helped us revert to earlier stages of development, review each other's work, and maintain a clean project history.

- **DVC (Data Version Control)** allowed us to handle large datasets and model files without storing them directly in Git. With DVC, we could version our data and model checkpoints, track changes, and reproduce training outputs by linking them to specific commits and configurations.

- **Docker** was used to create a consistent computing environment. It packaged our dependencies, Python version, and OS-level setup into a container, ensuring that the project could be run on any machine without worrying about compatibility or missing libraries.

- **GitHub Actions** was implemented to automate our Continuous Integration (CI) pipeline. Every time new code was pushed to the repository, it triggered workflows that automatically built the Docker image and ran the training pipeline, helping us verify that the system worked reliably at every step.

- **YAML Configuration Files** were used to make our experiments flexible and modular. Instead of hardcoding parameters, we stored them in `config.yaml`, allowing us to easily change input dimensions, learning rates, or dataset size without modifying the training code itself.

Together, these tools formed the backbone of our reproducible workflow. Each one contributed to a different layer of the project — from code versioning and data tracking to

automation and environment management — making our machine learning pipeline transparent, scalable, and easy to replicate.

4. Team Contributions and Project Roles

This project was a collaborative effort focused on making machine learning workflows reproducible, modular, and automated. The tasks were distributed logically among the team, with each member contributing to different aspects of the reproducibility process. Here's an overview of the major responsibilities and tasks completed as a team:

- **Data and Model Versioning**:
  The team initialized DVC in the project to manage datasets and model artifacts. The `data/` and `models/` directories were tracked using DVC, and a reproducible pipeline was defined in `dvc.yaml`. This ensured that every stage of the workflow—from training inputs to outputs—was version-controlled and traceable. Appropriate `.gitignore` and `.dvcignore` files were maintained to keep the repository clean and optimized.

- **Modular Code and Configuration**:
  The training and evaluation logic was implemented using clean, reusable Python modules. The code was structured across dedicated directories such as `datasets/`, `models/`, and `utils/`. The training script (`train.py`) was built to accept configurations from a YAML file, allowing flexibility for experiments and better parameter tuning. All experiment results were logged in structured formats to support analysis and reporting.

- **Environment and CI/CD Automation**:
  A `Dockerfile` was created to define a consistent environment for model training and testing. This made the setup portable and eliminated dependency issues across different systems. To further ensure reproducibility, a shell script (`run.sh`) was written to automate training steps inside the container. GitHub Actions was used to implement CI/CD pipelines that ran tests automatically whenever new code was pushed to the repository, verifying that the full workflow ran correctly from end to end.

This well-divided and cooperative approach allowed the team to efficiently build a reliable, reproducible ML project while gaining hands-on experience with industry-standard tools.

5. Implementation Overview

Our machine learning project was implemented using a modular and reproducible pipeline. The training process was controlled via a configuration file (`config.yaml`), allowing easy updates to parameters without changing the core code. The script `train.py` handled model training and saved results and outputs in structured directories.

To version data and models effectively, we used **DVC**. It tracked the `data/` and `models/` directories and defined a reproducible pipeline in `dvc.yaml`. This allowed us to link specific data and model versions to corresponding code changes, improving traceability.

We containerized the environment using **Docker**, ensuring that the project could run identically across systems. A shell script (`run.sh`) automated training inside the container.

To maintain automation and consistency, **GitHub Actions** was configured. It ran CI workflows on each code push to verify that the entire pipeline executed correctly.

Together, these tools allowed us to build a workflow that is not only functional but reproducible, portable, and easy to maintain.

6. Challenges Faced

- **Managing large datasets with Git (solved using DVC)**
  We initially tried storing dataset files directly in Git, which made the repository heavy and hard to manage. Switching to DVC allowed us to version data efficiently without bloating the repository.

- **Ensuring consistent results across systems (solved using Docker)**
  Differences in local environments caused compatibility issues. Docker helped us create a controlled setup where the code behaved the same on any system.

- **Tracking hyperparameter changes (solved using experiment logging)**
  Without a structured approach, it was difficult to track which configuration produced which result. Logging outputs and using `config.yaml` helped bring clarity and control.

- **Configuring GitHub Actions correctly for CI/CD**
  Setting up the CI workflow involved some trial and error. Paths, dependencies, and Docker commands had to be carefully aligned for the automation to work.

- **Structuring the DVC pipeline properly**
  Writing `dvc.yaml` with the right inputs, outputs, and command logic took some effort, but once completed, it streamlined our reproducibility process.

7. Best Practices We Followed

- Used modular Python code and external configuration for flexibility.

- Avoided pushing data/model files directly to Git by using DVC.

- Logged training metrics and saved models in structured directories.

- Documented the project clearly via `README.md` and `requirements.txt`.
- Used Docker for consistent environment setup across systems.
- Ensured CI/CD checks through GitHub Actions for every push.
- Maintained a clean and logical folder structure for collaboration.
- Used version control (Git) consistently to track changes in code and configuration.

8. Key Learnings and Takeaways

- Reproducibility is not just about repeating results — it's about building reliable, transparent workflows that others can understand and reuse.
- Using DVC taught us how to manage large datasets and models without cluttering the version control system.
- Docker helped us understand the importance of isolated environments and how they solve compatibility issues in ML workflows.
- GitHub Actions introduced us to CI/CD automation, making us aware of how even simple scripts can ensure code health and consistency.
- YAML-based configuration files improved our awareness of how to separate logic from parameters for clean experimentation.
- We learned the importance of documentation — a well-written README and properly named files made collaboration easier.
- As a team, we understood the value of dividing responsibilities and supporting each other's components for a smooth, integrated project.
- Team collaboration gave us experience in synchronizing tools like Git, DVC, and Docker within a shared repo, which mirrors real-world engineering teamwork.

9. Conclusion

Building a reproducible machine learning workflow is more than a technical task — it's a mindset. Throughout this project, we learned how structure, version control, and environment consistency form the foundation of trustworthy and scalable ML systems. By combining tools like Git, DVC, Docker, and GitHub Actions, we not only improved the reliability of our results but also prepared ourselves to work in real-world development environments where collaboration and reproducibility are essential.

Each part of the workflow — from tracking data to automating pipelines — was an opportunity to think critically, solve real implementation challenges, and build habits that align with professional best practices. Most importantly, this project taught us how thoughtful

planning and team coordination can transform simple experimentation into a robust and reusable solution.

We now leave this project with stronger technical skills, a deeper appreciation for workflow design, and a clear understanding of what it means to produce machine learning work that others can trust and build upon.

10. References

- **Git Documentation.** *Pro Git Book, 2nd Edition.* Available at: https://git-scm.com/book/en/v2

- **DVC Documentation.** *Data Version Control.* Available at: https://dvc.org/doc

- **Docker Documentation.** *Docker Overview.* Available at: https://docs.docker.com/get-started/overview/

- **GitHub Actions Documentation.** *Automating Your Workflow.* Available at: https://docs.github.com/actions

- **Goodman et al. (2016).** *What Does Research Reproducibility Mean? Science Translational Medicine,* 8(341), 341ps12. DOI: 10.1126/scitranslmed.aaf5027

- **MLflow: An Open Platform for the Machine Learning Lifecycle.** *Proceedings of the 2nd MLSys Conference,* 2019. Available at: https://www.mlflow.org/docs/latest/index.html

- **Pimentel et al. (2019).** *A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks. Proceedings of the 16th Mining Software Repositories Conference.* DOI: 10.1109/MSR.2019.00035