Software Engineer Report

Churn Predictor Web Application

Muhammad Mubashar Shahzad
Department of Mathematics and Geosciences
Course: Scientific and Data Intensive Computing
University of Trieste
Email: mshahzad@sissa.it

I. Problem Definition

The objective of this project is to design and implement a reproducible and modular MLOps pipeline to predict customer churn using structured banking data. The dataset, obtained from Kaggle (*Credit Card Customers*), contains customer demographic information, transactional behavior, and credit profiles.

From the perspective of a Software Engineer, the core challenge lies in establishing a maintainable and automated software architecture that supports:

- Designing a modular and reusable codebase for data preprocessing, model training, and evaluation.
- Enabling continuous integration and continuous deployment (CI/CD) workflows through GitHub Actions.
- Ensuring reproducibility via environment management and dependency tracking (e.g., `requirements.txt`).
- Implementing data version control using DVC to manage input datasets and processed artifacts.
- Utilizing containerization through Docker to maintain consistency across development and deployment environments.

The project involves seamless collaboration between roles-Data Engineer, Software Engineer, and Software Developer-within a traceable, testable, and version-controlled pipeline that facilitates efficient machine learning experimentation and deployment readiness.

II. Introduction

Customer churn, the phenomenon where customers stop using a companys services, poses a significant challenge for businesses across various industries. Predicting which customers are likely to churn allows organizations to take proactive measures to retain valuable clients, thus reducing revenue loss and improving customer lifetime value [1], [2]. Machine learning models have shown strong capabilities in identifying churn patterns based on historical customer data [3]. However, to effectively use these models, businesses require reliable software systems that expose predictions through user-friendly applications.

This report focuses on the software engineering aspects of deploying a churn prediction model as a scalable and secure backend service. The objective is to create an API that integrates seamlessly with front-end applications, ensuring low latency, high availability, and data security.

By delivering real-time churn predictions accessible via a web application, this system enables marketing and customer success teams to make informed decisions, ultimately contributing to improved customer retention and business growth.

III. Software Development Method and Process Model

In this project, we adopt the **Waterfall model** as the software process model. Unlike Agile, which emphasizes iterative development and frequent cross-role collaboration, the Waterfall approach aligns better with our academic constraints and team structure. The project follows a sequential flow where tasks are clearly divided across three roles:

- **Data Engineer (Vishal Nigam)**: Handles data ingestion, preprocessing, model training, and versioning tasks.
- **Software Engineer (Muhammad Mubashar Shahzad)**: Implements testing strategies, CI/CD pipelines, Docker-based reproducibility, and ensures adherence to MLOps standards.
- **Software Developer (Sepehr)**: Focuses on software API development and frontend deployment of the model (e.g., Streamlit app).

The development methodology includes:

- Designing a modular folder structure to maintain clear responsibility boundaries (e.g., `src/`, `models/`, `results/`).
- Using Git for source code versioning and GitHub for collaborative integration.
- Integrating Continuous Integration (CI) using GitHub Actions to run automated unit tests and code linting on every push or pull request.
- Enabling Continuous Deployment (CD) readiness by containerizing the entire project using Docker.

The Waterfall process ensures that each component is completed, validated, and documented before the next stage begins. This disciplined approach makes responsibilities clear, simplifies debugging, and supports consistent handover between roles.

IV. Software Requirements and Development Methods

A. 4.1 Software Requirements

Functional Requirements:

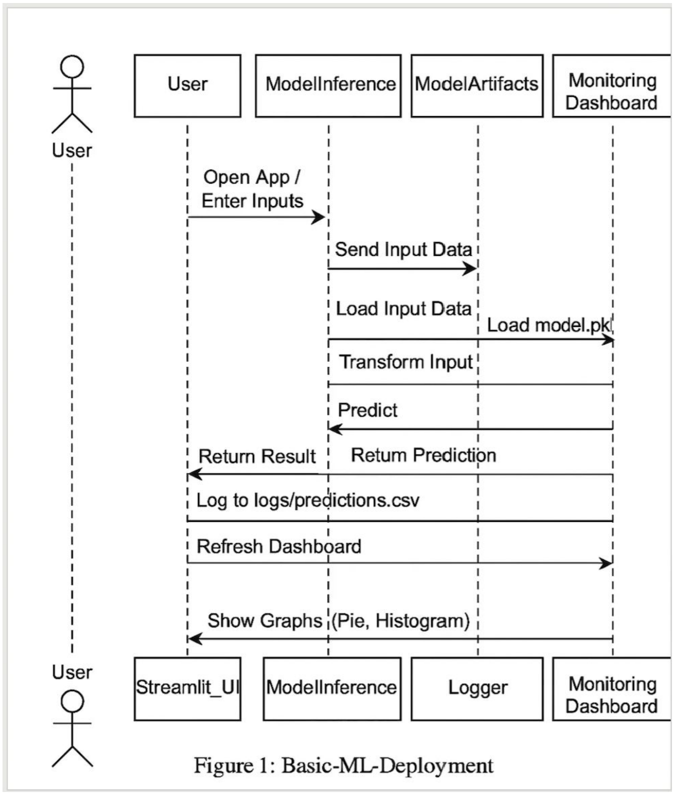

Fig. 1: Component Diagram of the MLOps Pipeline

- The system accepts a bank customers input features and returns a prediction for churn likelihood.
- The system allows retraining the model with new datasets.
- It logs model predictions and processing steps to ensure traceability.

Non-Functional Requirements:

- **Reproducibility**: All experiments and preprocessing steps remain fully reproducible.
- **Maintainability**: The codebase stays modular and easy to extend or refactor.
- **Scalability**: Model versioning and containerization support future scaling needs.
- **Version Control**: All code, data, and models are tracked using Git and DVC.

B. *4.2 Software Development Methods*

Version Control Strategy:

- We adopt a `feature-branch  main` Git strategy to manage development.
- Git tags (e.g., `v1.0`) mark milestone releases of the pipeline.
- Data files and model binaries are versioned using **DVC** to preserve dataset integrity across environments.

Environment Management:

- All the project dependencies are listed in the `requirements.txt`.
- The project runs inside a `venv` environment for local development, and within a `Docker` container for deployment purposes.

C. *4.3 Software Testing*

Unit Testing:

- We implement tests using the `pytest` framework within the `tests/` directory.
- Test cases verify correct data preprocessing and validate the shape and structure of inputs and outputs.

CI Integration:

- The `GitHub Actions` are configured throught the `.github/workflows/test.yml` to automate:
  - Unit testing with `pytest`
  - Code linting using `flake8`
  - DVC status checks to ensure data pipeline consistency

This robust software engineering foundation ensures the system remains reliable, repeatable, and easy to maintain as it evolves.

V. MLOps Approaches and Issues

A. *5.1 MLOps Practices Applied*

To ensure a robust and production-ready machine learning workflow, we apply the following MLOps principles:

- **Version Control:** Git manages source code versioning, while DVC tracks dataset versions and pipeline artifacts.
- **Reproducibility:** Models and preprocessing pipelines are saved and versioned with timestamps. The environment is controlled via `requirements.txt` and Docker.
- **Experiment Management:** Although MLflow is not used due to academic constraints, model metadata such as training time, parameters, and scores are logged in `best_model_meta.json` for transparency and reproducibility.
- **CI/CD Pipeline:** GitHub Actions automate:
  - Unit tests using `pytest`
  - Code linting via `flake8`
  - DVC pipeline integrity checks
- **Dockerization:** A `Dockerfile` packages the pipeline for isolated deployment. The final pipeline runs successfully using Docker with a mounted volume setup.

B. *5.2 Issues Faced and Resolved*

- **CI Failures:** Initial GitHub Actions runs fail due to missing datasets and incorrect `PYTHONPATH`. We resolve this by mocking minimal data and setting `PYTHONPATH=.` in the workflow file.
- **DVC Compatibility:** The Docker container requires correct mounting of version-controlled data from the host system to resolve `FileNotFoundError`.
- **Testing Failures:** Unit tests initially fail due to incomplete mock datasets and insufficient class distribution. We address this by adapting the tests to validate functionality with minimal representative data.

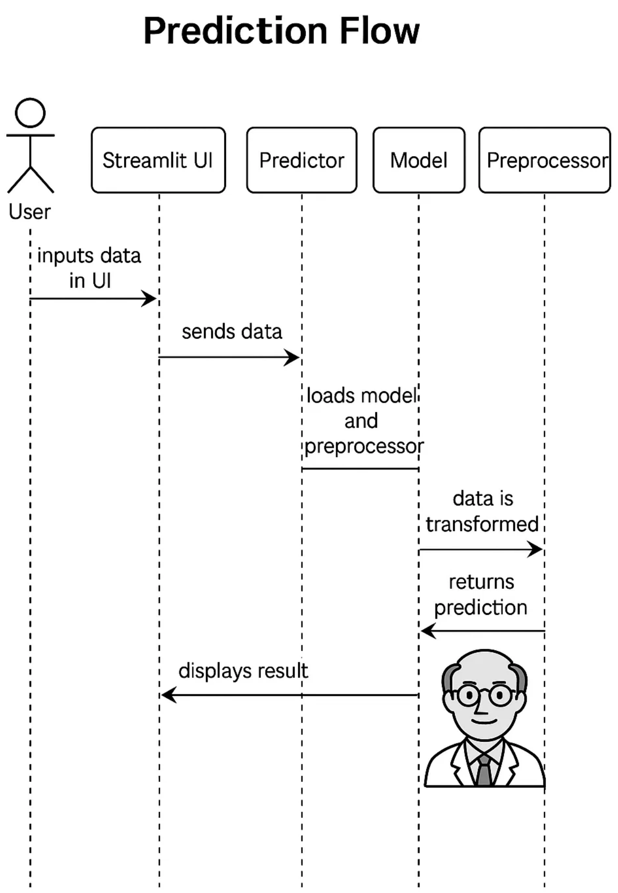

Fig. 2: Sequence Diagram of Prediction Flow from UI to Model Output

Summary: Despite infrastructure and reproducibility limitations in an academic environment, the applied MLOps practices ensure modular development, automated quality checks, and consistent execution across environments.

VI. Proposal Limitations

While the proposed MLOps pipeline demonstrates strong modularity, automation, and reproducibility, several limitations emerge due to the academic context and resource constraints:

- **Limited Experiment Tracking:** Due to infrastructure constraints, we do not integrate tools like MLflow or Weights & Biases. As a result, model tracking relies on manually logged metadata in JSON format.
- **No External Deployment Platform:** The final Streamlit application is not deployed on platforms like Streamlit Cloud or Heroku. Testing and usage remain confined to local environments or Docker-based setups.
- **Simplified Test Coverage:** While unit tests validate core functionality, they are not exhaustive. Full test coverage for edge cases, integration scenarios, and performance metrics is currently lacking.
- **Dataset Constraints:** The dataset used is static and pre-cleaned. Real-world production systems typically require continuous ingestion of new data, which is not implemented in this prototype.
- **Absence of CI/CD Deployment Stage:** Although CI for testing and linting is in place, we do not configure full continuous deployment to production due to academic scope limitations.

These limitations reflect practical trade-offs made to meet academic goals while still demonstrating core MLOps competencies. In a real-world setting, addressing these issues would significantly enhance scalability, maintainability, and deployment readiness.

VII. Conclusion

This project successfully demonstrates the application of core MLOps principles to a real-world problem: predicting customer churn from structured banking data. As the Software Engineer, I ensure that the system is modular, reproducible, and aligned with best practices in software development and machine learning operations.

The pipeline integrates source control (Git), data versioning (DVC), environment reproducibility (Docker and `requirements.txt`), and automation (GitHub Actions for CI). Despite limitations in deployment and experiment tracking due to academic constraints, the implemented solution supports seamless collaboration, clean handoffs between roles, and reliable model execution in controlled environments.

The MLOps infrastructure developed lays the groundwork for future scalability, enabling future enhancements such as continuous data ingestion, advanced model tracking (e.g., MLflow), and cloud-based deployment platforms. Overall, the project highlights the importance of engineering discipline in building robust and maintainable machine learning systems.

VIII. Future Work

While the current implementation meets the foundational goals of modularity, reproducibility, and automation, several avenues exist to enhance and extend the system:

- **Advanced Experiment Tracking:** Integrating tools like MLflow or Weights & Biases would enable detailed experiment logging, parameter tracking, and comparison dashboards for model performance over time.
- **Cloud Deployment:** Deploying the Streamlit application on a cloud platform (e.g., Streamlit Cloud, Heroku, or AWS EC2) would make the system accessible to non-technical users and stakeholders.
- **Automated Data Pipelines:** Implementing ingestion pipelines for real-time or batch updates of new customer data would allow dynamic retraining and monitoring.
- **Model Monitoring and Drift Detection:** Adding live monitoring dashboards and alert systems to detect model performance degradation or data distribution drift would increase reliability in production.

- **Security and Authentication:** For production use, access control, user authentication, and secure API endpoints would be necessary to protect sensitive data and services.
- **Expanded Test Coverage:** Increasing unit and integration test coverage would improve robustness and reliability, especially for edge cases and high-volume data scenarios.

These improvements would elevate the prototype to a fully production-grade MLOps pipeline capable of supporting continuous learning, high availability, and enterprise-level compliance.

References

[1] Alotaibi, M. Z., & Haq, M. A. (2024). Customer churn prediction for telecommunication companies using machine learning and ensemble methods. *Engineering, Technology & Applied Science Research, 14*(3), 14572-14578.

[2] Jamalian, A., & Fokurdi, R. (2014). Application of Data Mining Techniques in Customer Churn Management. *In Second National Conference on Applied Research* (p. 1).

[3] Wang, X., Nguyen, K., & Nguyen, B. P. (2020, January). Churn prediction using ensemble learning. *In Proceedings of the 4th international conference on machine learning and soft computing* (pp. 56-60).