**MLOps Pipeline for NLP Text Classification**

**Abdullah Latif 21i-2506**

**Muhammad Hussain 21i-0501**

**1. Introduction**

This project outlines the development and implementation of an end-to-end Machine Learning Operations (MLOps) pipeline designed for an NLP text classification task. The primary objective was to apply core MLOps principles to automate, streamline, and monitor the lifecycle of a machine learning model, from data ingestion to production serving. The chosen task involves classifying news headlines from the "News Category Dataset" into their respective categories, demonstrating a practical application of MLOps in the NLP domain.

**2. System Architecture**

The pipeline is structured around key MLOps stages, ensuring a modular and reproducible workflow:

1. **Data Pipeline (Apache Airflow):** Responsible for automated data ingestion from the source, comprehensive preprocessing tailored for NLP tasks (including cleaning, lemmatization, and basic feature engineering like text length), and splitting the data into training, validation, and test sets.
2. **Model Experimentation & Management (MLflow):** Facilitates the training of multiple machine learning models. MLflow is used for meticulous tracking of experiments (parameters, code versions, metrics like F1-score and accuracy) and for versioning and managing trained models via the MLflow Model Registry.
3. **Model Serving (FastAPI):** A high-performance REST API built with FastAPI loads a selected model version (e.g., designated by a "Production" alias) from the MLflow Model Registry. It exposes endpoints for real-time single and batch predictions, along with model metadata.
4. **Monitoring (Prometheus & Grafana):** The FastAPI application is instrumented to expose key operational and model-specific metrics. Prometheus scrapes these metrics, and Grafana dashboards are used for visualization, enabling monitoring of API performance, prediction behavior, and potential data drift.

Containerization (Docker, Docker Compose) is utilized for Airflow, Prometheus, and Grafana to ensure consistent environments and simplify deployment.

**3. Data Pipeline: Apache Airflow**

The data pipeline, orchestrated by Apache Airflow, forms the foundation for reproducible model training.

* **Dataset:** The "News Category Dataset" from Kaggle was used, containing over 200,000 news headlines across 42 categories. Headlines and short descriptions were combined for textual input. A 10% sample was used during development for efficiency.
* **DAG (**news\_category\_data\_processing\_v1**):**
  + setup\_directories\_task: Ensures output directories exist.
  + ingest\_data\_task: Loads, combines text fields, samples data, and saves the initial dataset.
  + basic\_cleaning\_task: Converts text to lowercase, removes punctuation and numbers.
  + advanced\_processing\_task: Performs stopword removal and NLTK-based lemmatization.
  + feature\_engineering\_task: Calculates text length and a placeholder sentiment score.
  + split\_data\_task: Stratified split into train, validation, and test sets.
* **Orchestration:** The DAG is scheduled to run daily at 2:00 AM UTC (0 2 \* \* \*), with catchup=False. Airflow services run within Docker containers managed by Docker Compose. Airflow Variables were used for configurable parameters like random seed and split sizes.

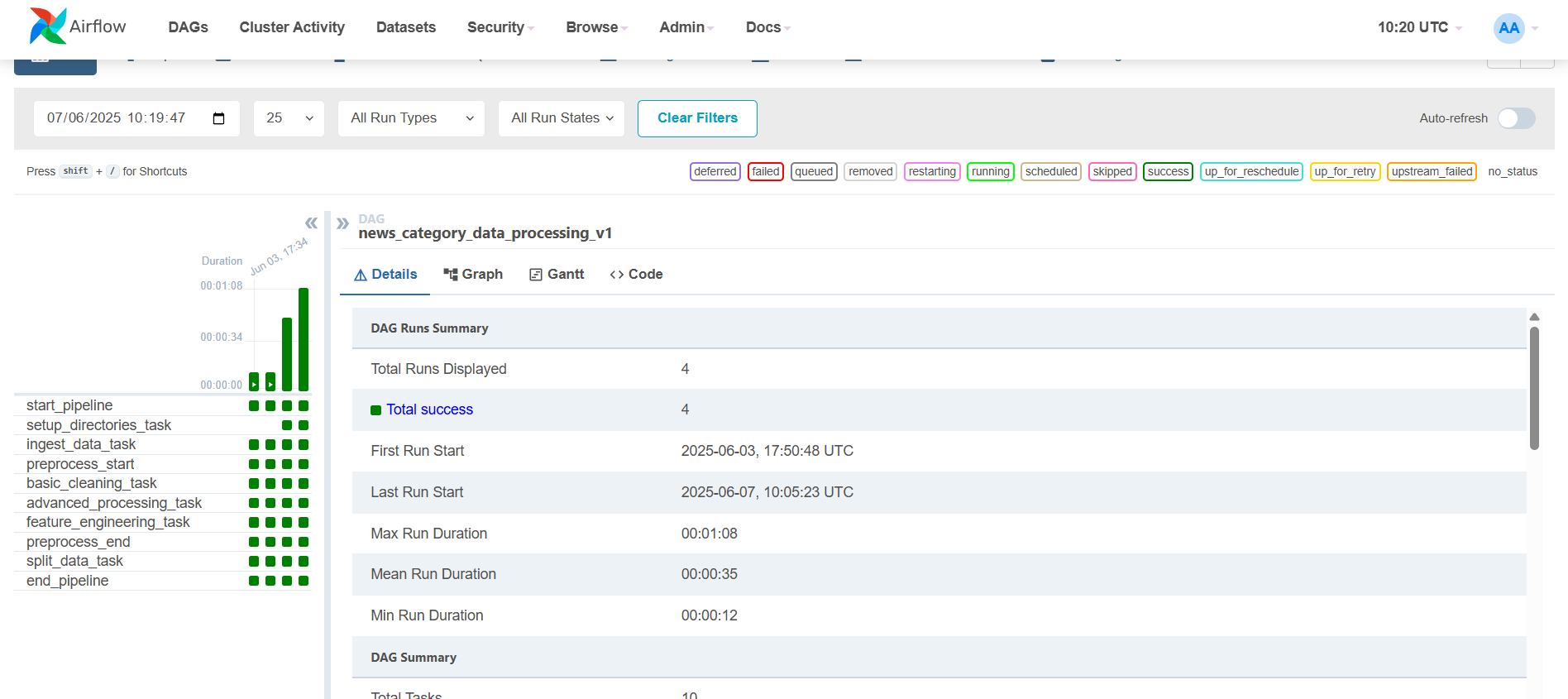


Figure 1: Airflow DAG for the news data processing pipeline.

**4. Model Development and Experimentation: MLflow**

A systematic approach was taken for model development, with MLflow serving as the central tool for experiment management.

* **Models Explored:**
  1. **TF-IDF + Multinomial Naive Bayes:** Baseline traditional ML model.
  2. **TF-IDF + Logistic Regression:** Another robust traditional ML baseline.
  3. **Keras Neural Network (Custom Embeddings):** Simple NN with a trainable Embedding layer.
  4. **Keras Neural Network (GloVe Embeddings):** NN utilizing pre-trained GloVe word vectors (100d).
  5. **DistilBERT (Fine-tuned):** A transformer-based model, fine-tuned on a sample of the data for sequence classification.
* **Experiment Tracking:** For each run, MLflow logged:
  1. **Parameters:** Key hyperparameters (e.g., TF-IDF max\_features, NN embedding\_dim, epochs, batch\_size, transformer max\_length).
  2. **Metrics:** Validation accuracy, weighted F1-score, training time, and inference time.
  3. **Artifacts:** Serialized models, tokenizers/vectorizers, and the label encoder.
* **Model Selection & Registry:** Based on validation F1-scores and considering computational cost, the **Logistic Regression model (F1-score:**0.4513 was chosen as the primary candidate for serving. This model version was registered in the MLflow Model Registry as NewsCategoryClassifier and assigned the alias Production.

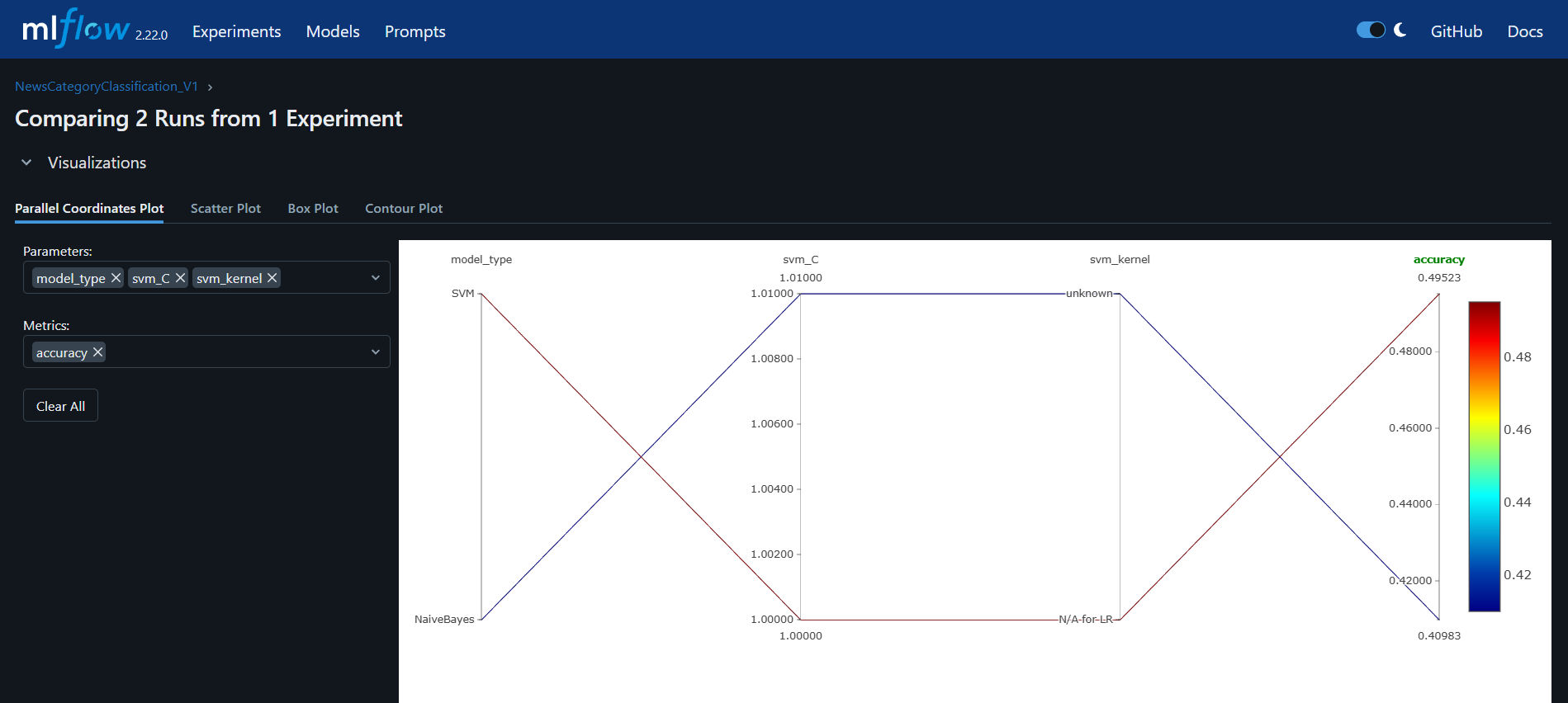


Figure 2: MLflow UI showing model experiment comparison.

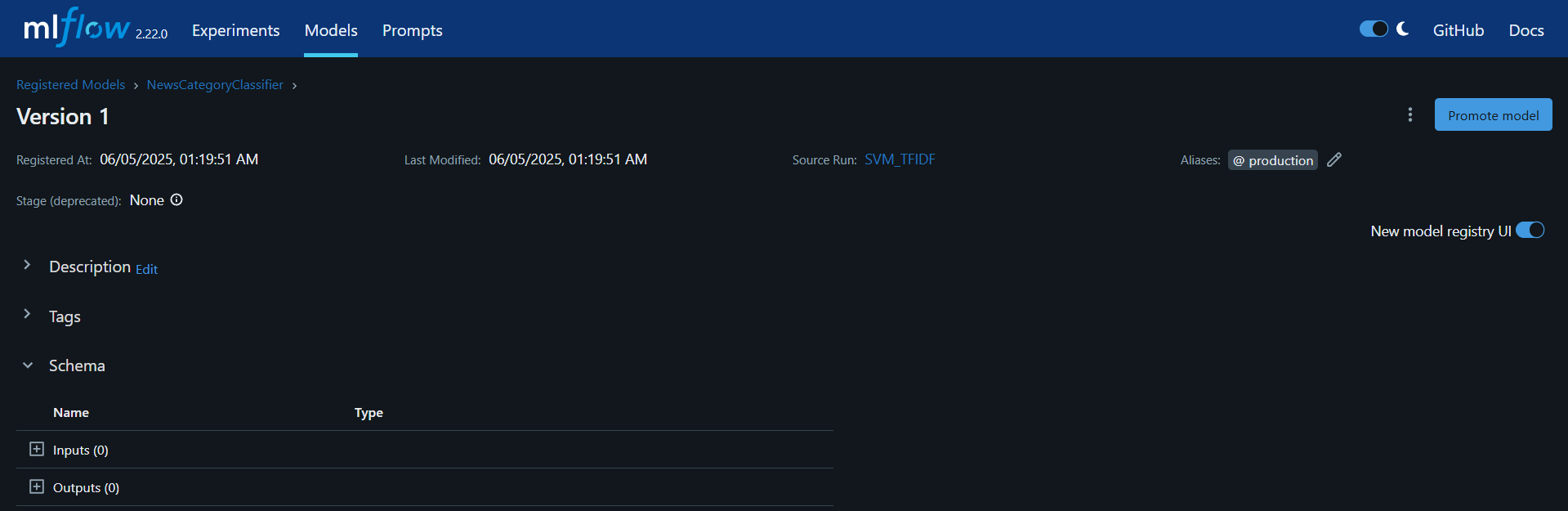
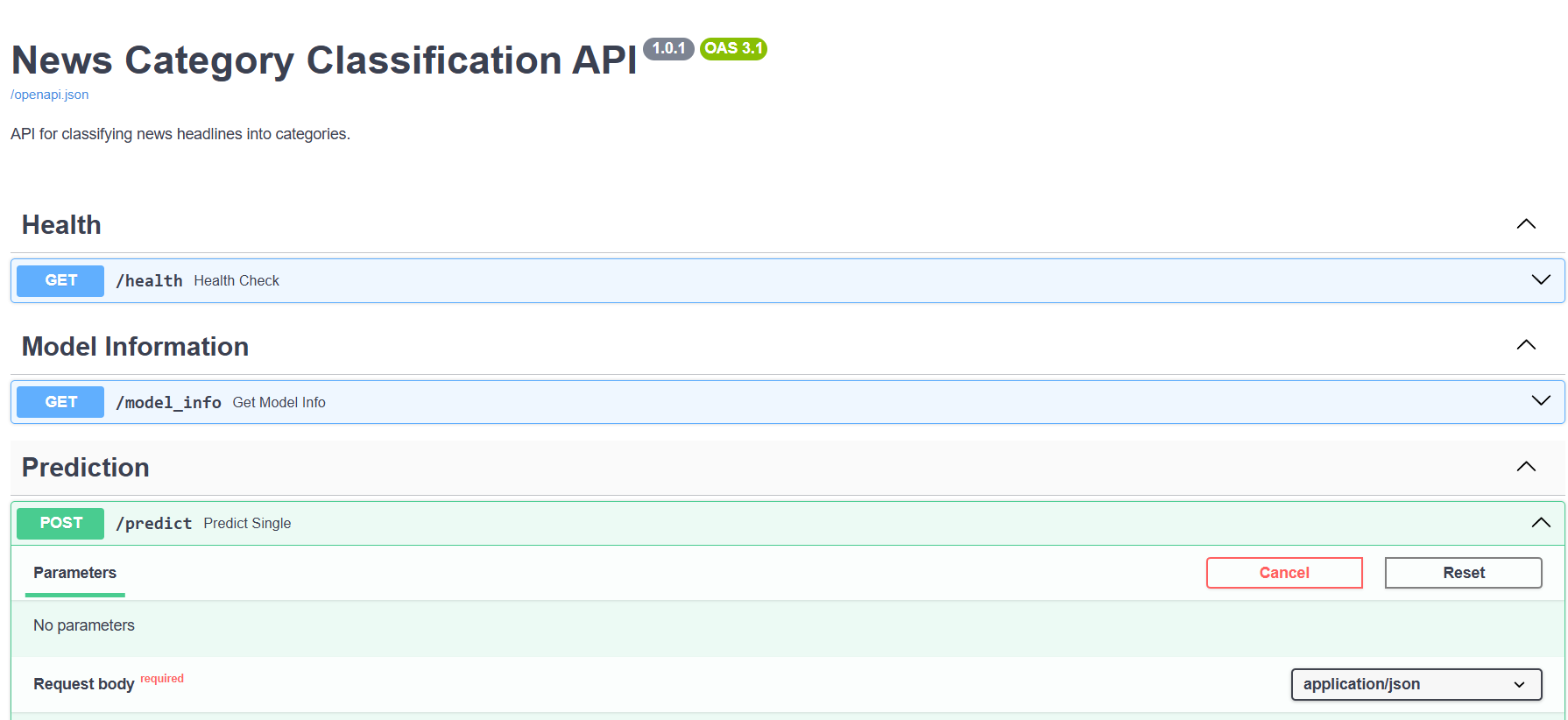


Figure 3: NewsCategoryClassifier in MLflow Model Registry with 'Production' alias.

**5. Model Serving: FastAPI**

A RESTful API was developed using FastAPI to serve the selected model for predictions.

* **API Endpoints:**
  + GET /health: Checks API and model loading status.
  + GET /model\_info: Provides metadata of the loaded model (name, version, alias, metrics).
  + POST /predict: Classifies a single input text.
  + POST /predict\_batch: Classifies a batch of input texts.
* **Model Loading:** The API loads the model version designated by the Production alias (configurable via MODEL\_ALIAS env var) from the MLflow Model Registry upon startup. It dynamically loads the appropriate preprocessors based on the model flavor inferred from MLflow artifacts.
* **Validation & Documentation:** Pydantic models enforce request/response schemas. FastAPI automatically generates OpenAPI documentation, accessible via /docs (Swagger UI) and /redoc.
* **Web Demo:** A basic api\_demo.html page was created for easy interaction with the API. CORS middleware was enabled in FastAPI.



**Caption:** Figure 4: Swagger UI for the deployed FastAPI application.

**6. Monitoring: Prometheus & Grafana**

To observe the API and model in a simulated production setting, a monitoring stack was implemented.

* **API Instrumentation:** The FastAPI application was instrumented using prometheus-fastapi-instrumentator. This exposed a /metrics endpoint with standard metrics (request count, latency, errors) and custom metrics:
  + api\_prediction\_category\_total: Counts predictions per category.
  + api\_input\_text\_length\_chars: Histogram of input text lengths (proxy for data drift).
  + api\_custom\_error\_total: Counts specific application-level errors.
* **Prometheus:** A Dockerized Prometheus instance scrapes the /metrics endpoint of the API at regular intervals (15s).
* **Grafana:** A Dockerized Grafana instance uses Prometheus as a data source. A custom dashboard was created to visualize:
  + **API Performance:** Request rate, P95 latency.
  + **Prediction Behavior:** Distribution of predicted categories.
  + **Data Characteristics:** Average input text length.
  + **Error Monitoring:** HTTP 5xx error rates and custom application error counts.

**7. Key Challenges & Learnings**

* **Challenges Encountered:**
  + **Git Large File Management:** Initial difficulties pushing large dataset and embedding files to GitHub required learning and applying .gitignore best practices and Git history rewriting techniques (interactive rebase, and ultimately git filter-repo for a clean slate).
  + **Cross-Component Integration:** Ensuring seamless operation between Dockerized services (Airflow, Prometheus, Grafana) and locally run components (MLflow server, FastAPI during dev) involved careful configuration of networking, paths, and environment variables.
  + **Resource Constraints:** Training advanced models like DistilBERT on local hardware necessitated data sampling and reduced training epochs, highlighting the need for scalable compute resources for larger NLP tasks.
* **Key Learnings:**
  + Gained a comprehensive, practical understanding of the entire MLOps lifecycle.
  + Recognized the critical role of automation (Airflow) for data pipelines and reproducibility.
  + Appreciated the power of MLflow for organized experiment tracking, model comparison, and version control.
  + Developed skills in building and deploying robust, documented APIs using FastAPI.
  + Understood the importance of continuous monitoring (Prometheus, Grafana) for maintaining model and application health in production.
  + Reinforced the benefits of containerization for creating consistent and portable environments.

**8. Conclusion**  
This project successfully demonstrated the design and implementation of a functional end-to-end MLOps pipeline for an NLP text classification task. By integrating tools like Airflow, MLflow, FastAPI, Prometheus, and Grafana, the pipeline automates critical stages from data preparation to model serving and monitoring. This hands-on experience provided valuable insights into the principles and practices required for operationalizing machine learning models effectively.