Promptly - Model Development Report

---

1. Overview

This report documents the model development process for *Promptly*, an AI-powered document-based Q&A system. The model development focuses on loading preprocessed embeddings from the data pipeline, preparing for model training, fine-tuning large language models (LLMs), model validation, sensitivity analysis, bias detection, and pipeline automation for continuous improvement.

---

2. Model Development Process

2.1 Loading Data from the Data Pipeline

The Promptly data pipeline processes user-uploaded PDFs and text files through PII detection, semantic chunking, and embedding generation. Embeddings are stored in Supabase using pgvector and versioned via DVC and GCS. The model pipeline loads this data via `load_data.py` using both local versioned CSVs and live Supabase queries.

2.2 Model Training & Selection

We trained multiple classification models — Logistic Regression, SVM, and k-NN — for query category prediction.

- **Embeddings:** Generated with `nomic-embed-text-v1.5`.
- **Training:** Conducted using Leave-One-Out Cross-Validation due to limited dataset size.
- **Model Selection:** The best model was chosen based on accuracy and stability across validation runs.

2.3 Model Validation

Validation metrics included:

- Accuracy
- Precision

- Recall
- F1-score
  Validation was performed on hold-out sets and confirmed that k-NN with 3 neighbors performed the best.

2.4 Bias Detection

We implemented bias detection by measuring user dominance in the training data.

- **Method:** Aggregating chunk contributions by `upload_user_id` using Supabase join queries.
- **Threshold:** If any single user contributes more than 50% of document chunks, retraining is blocked.
- **Implementation:** Integrated into Airflow retraining DAG using a branch operator.
  Future plans include slicing-based bias detection by document category.

2.5 CI/CD Pipeline Integration

- An Airflow DAG automates retraining, with bias detection gating.
- Model training and registry push steps are containerized and ready for GitHub Actions deployment triggers.

---

3. Hyperparameter Tuning

Hyperparameters tuned:

- **k-NN:** Explored neighbors between 1–5; `n_neighbors=3` yielded the best result.
- **SVM:** Used default `liblinear` solver with auto gamma selection.
  Hyperparameter decisions are tracked in experiment logs.

---

4. Experiment Tracking & Results

- Used MLflow for experiment tracking (under setup).
- Validation results were compared visually with confusion matrices (attached in the appendix).
- Results table:

Model	**Accuracy (LOO-CV)**
Logistic Regression	82%
SVM	80%
k-NN (n=3)	85%

---

5. Model Sensitivity Analysis

- **Feature Importance:** Not applicable (dense embedding vectors).
- **Hyperparameter Sensitivity:** Minor fluctuations were noted when tuning `k` in k-NN; best performance stable at `n=3`.

---

6. Model Bias Detection (Using Slicing Techniques)

- Current implementation: User dominance bias detection via join-based analysis of `document_chunks` and `documents` in Supabase.
- Future roadmap:
    - Slice performance analysis by document categories.
    - Visual bias reports via Fairlearn and TFMA once larger datasets are available.
    - Planned bias mitigation through re-sampling and balanced synthetic data generation.

---

7. CI/CD Pipeline Automation for Model Development

- **Airflow Orchestration:** DAG with integrated bias detection gating.
- **Trigger Setup:** Planned GitHub Actions integration.
- **Validation & Metrics Logging:** Automatic metrics computation post-training.
- **Automated Bias Detection:** Included in the DAG branch logic.
- **Model Registry Push:** Planned integration with GCP Artifact Registry.
- **Rollback Mechanism:** If the new model underperforms, revert to the last stable model pickle.

---

8. Code Implementation Summary

- **Data Loading:** `load_data.py` integrated with Supabase and versioned data sources.
- **Training & Selection:** `train_model.py` handles training, validation, and saving of best models.
- **Bias Detection:** `bias_detection.py` fully integrated with Supabase and Airflow.
- **Inference Service:** FastAPI-based inference service (`inference_service.py`) using FAISS and fallback LLMs.
- **Front-end:** Streamlit app for end-user queries.
- **Containerization:** Dockerfile (WIP) for reproducibility and deployment.
- **Monitoring:** Planned W&B and Prometheus integration for long-term tracking.

---

9. Model Fine-Tuning and Synthetic Data Generation Pipeline

9.1 Fine-Tuning Qwen2.5-0.5B-Instruct

- **Base Model:** `Qwen/Qwen2.5-0.5B-Instruct`
- **Fine-Tuning Framework:** LoRA adaptation (`r=16`, `lora_alpha=32`)
- **Dataset:** 69 conversational examples from the Supabase conversations table.
- **Training Config:**
    - Batch size: 1 (gradient accumulation steps = 8)
    - Learning rate: 2e-4
    - Epochs: 3
    - Tracked on W&B
- **Deployment:** Hosted on Hugging Face Hub: [rajiv8197/promptly-tuned]

9.2 Synthetic Data Generation Pipeline

- **Workflow:**
    - Ingest and redact documents
    - Semantic chunking and embedding generation
    - Generate queries and responses using LLaMA 3.2 8B
    - Store conversations in Supabase for future fine-tuning
- **Technology Stack:**
    - Data Pipeline: Airflow
    - Storage: Supabase
    - Embeddings: Nomic
    - Synthetic Generation: LLaMA 3.2 8B

9.3 Fine-Tuning Results

Example ID	Baseline ROUGE-L	Fine-Tuned ROUGE-L
0	0.5244	0.5029
1	0.8397	0.8730
2	0.7629	0.7957
3	0.4828	0.7955
4	0.7143	0.8434
5	0.5656	0.8681
6	0.2342	0.3700
7	0.8163	0.9023
Average	**0.6175**	**0.7438**

9.4 Future Improvements

- Expand the dataset with real user queries
- Add multi-modal document support
- Improve conversational diversity in synthetic data

---

10. Conclusion

The Promptly project has successfully established a robust end-to-end model pipeline integrating data ingestion, embeddings, classification, fine-tuning, and inference. Bias detection, model selection, and planned CI/CD automation ensure sustainable model development. Future improvements will focus on scaling, enhancing fairness detection, and automating deployment with GCP services.