BRIEF REPORT

Enhanced Text Analysis API Project

15/03/2025

Design decisions

The project implements a modern, containerized text analysis service following best practices in software architecture and MLOps. Key design decisions include:

API Architecture	FastAPI provides performance, automatic OpenAPI documentation, and built-in validation through Pydantic models. The API endpoints follow RESTful principles with clear request/response contracts.
Separation of Concerns	The codebase is organized into distinct modules ensuring maintainability and testability. This modular approach allows independent development of components.
Caching Strategy	A file-based request caching system prevents redundant model calls, significantly improving response times for repeated queries while reducing computational costs.
Containerization	Docker deployment enables consistent environments across development and production.
Error handling	Comprehensive error handling helps to understand and resolve problems. All errors are properly logged and tracked.
Configuration Management	Environment variables with defaults and validation ensure the application works correctly across different environments while keeping sensitive information secure.

Comparison of ML vs LLM approaches

The project combines both traditional ML and LLM approaches, offering insights into their relative strengths:

Performance	The traditional ML model (TF-IDF with LogisticRegression) processes requests in milliseconds, while LLM requests typically take 1-3 seconds. This performance gap makes the ML model preferable for high-volume, latency-sensitive applications.
Accuracy & Context Understanding	The LLM demonstrates superior understanding of context, nuance, and complex sentiment, particularly for ambiguous or mixed sentiment texts. The traditional ML model excels at classifying clear-

	cut examples but struggles with subtlety.
Resource Requirements	The traditional ML model requires minimal computational resources (< 100MB memory) and can run efficiently on CPU. LLMs require API calls to external services with associated costs and potential rate limitations.
Complexity	The traditional ML pipeline requires more data preprocessing and feature engineering, while LLM implementation is simpler but requires prompt engineering.
Output Richness	LLMs may provide richer outputs including explanations and entity extraction, while the traditional model offers only classification and confidence scores.

Potential improvements

A few improvements can bring this project up to production level:

Model improvement	Fine tuning a smaller language model could be better then current solution. Adding new dataset for current ML model will improve accuracy. Using ensemble methods that combine ML and LLM results could improve accuracy.
Infrastructure improvements	Implementing a more robust distributed caching system (Redis). Setting up proper monitoring with Prometheus and Grafana would improve reliability.
Advanced Analytics	Integrating feedback loops to track model performance over time,creating a retraining pipeline will ensure model accuracy.
Feature Expansion	Adding batch processing capabilities, streaming responses for LLM calls.
Security enhancements	Adding rate limiting, implementing proper authentication/authorization, and improving input validation will protect against potential attacks and abuse.