End-to-End MLOps Pipeline for Music Recommendation: A Production-Ready Implementation

STT890 Machine Learning Operations
Author: Sharod Williams and Siddhish Nirgude
Date: December 10, 2025

1. Background and Motivation

1.1 Problem Statement

The music streaming industry faces significant challenges in delivering personalized recommendations at scale. The Last.fm-360K dataset, containing interactions from 358,622 users across 126,442 artists with over 17 million listening events, exemplifies the extreme sparsity inherent in music recommendation systems (over 99.9% of user-artist pairs have zero interactions). Traditional recommendation approaches struggle with the cold start problem, where new users or artists lack sufficient interaction history for accurate predictions. Furthermore, user preferences evolve rapidly as musical trends shift, requiring systems that can adapt while maintaining computational efficiency and reliability.

The business value of effective recommendations extends beyond user satisfaction to include increased engagement, longer session times, and reduced churn rates. However, delivering these recommendations requires more than sophisticated algorithms; it demands robust production infrastructure capable of handling real-time requests, monitoring system health, detecting performance degradation, and managing failures gracefully. The gap between research prototypes and production-ready systems represents a critical challenge in modern machine learning operations.

1.2 Project Objectives

This project develops an end-to-end MLOps pipeline demonstrating the complete machine learning lifecycle from data ingestion through production deployment and monitoring. The system implements Alternating Least Squares (ALS) collaborative filtering, chosen for its proven effectiveness with implicit feedback datasets, computational scalability through parallelization, and interpretability via latent factors. The emphasis is placed on production readiness rather than algorithmic complexity, reflecting real-world priorities where reliable deployment often matters more than marginal accuracy improvements.

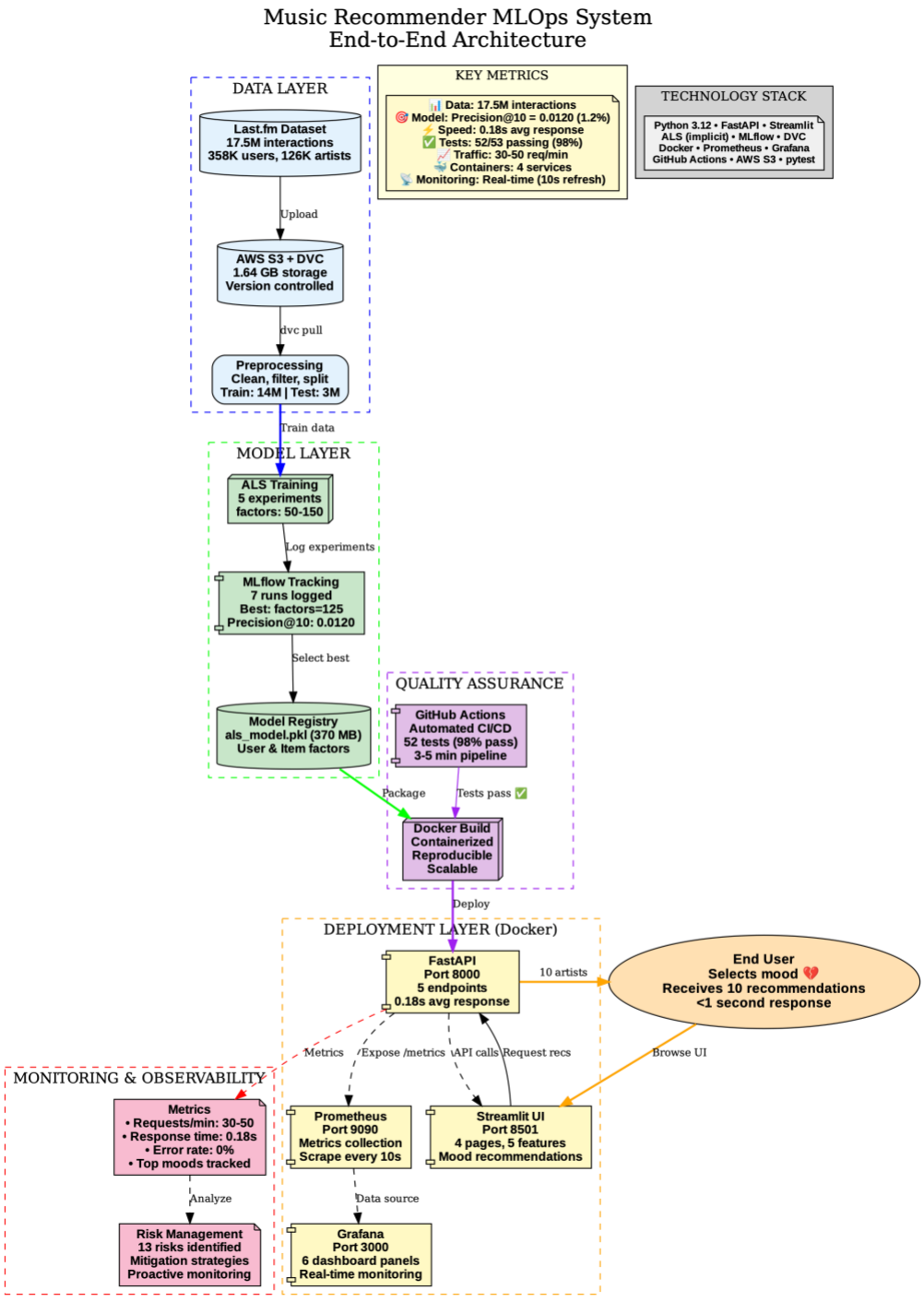
Key objectives align with the STT890 course requirements: (1) build a comprehensive MLOps pipeline encompassing data version control via DVC, experiment tracking through MLflow, automated testing with pytest, containerized deployment using Docker, and real-time monitoring with Prometheus and Grafana; (2) deploy a production-ready system accessible to non-technical users through an intuitive Streamlit interface; (3) implement robust monitoring infrastructure tracking system health, model performance, and user behavior; (4) address all seven project rubric requirements including process documentation, online data ingestion, data and model repositories, predictive modeling, user-accessible deployment, monitoring dashboards, and comprehensive risk documentation; and (5) demonstrate MLOps best practices including version control, continuous integration, automated testing, graceful degradation, and incident response capabilities.

1.3 Dataset and Approach

The Last.fm-360K dataset provides implicit feedback through user-artist play counts, representing realistic streaming behavior where users do not explicitly rate content. ALS matrix factorization decomposes the user-item interaction matrix into two lower-dimensional matrices representing latent user preferences and item characteristics. This approach offers computational efficiency, interpretability, and proven performance on sparse datasets.

Data preprocessing implements quality filters to ensure robust model training: users with fewer than five total interactions are removed to eliminate noise from casual listeners, and artists with fewer than three listeners are filtered to focus on established content. The confidence weighting parameter (alpha equals 40) transforms raw play counts into implicit confidence scores following the formula: confidence equals 1 plus 40 times play count. This distinguishes between strong preferences (many plays) and moderate interest (few plays). The train-test split employs an 80-20 temporal split per user, simulating realistic deployment where the model must predict future interactions. The complete pipeline processes 1.64 GB of raw data into a 14 GB sparse matrix representation.

Figure1: Master System Architecture



2. Methods

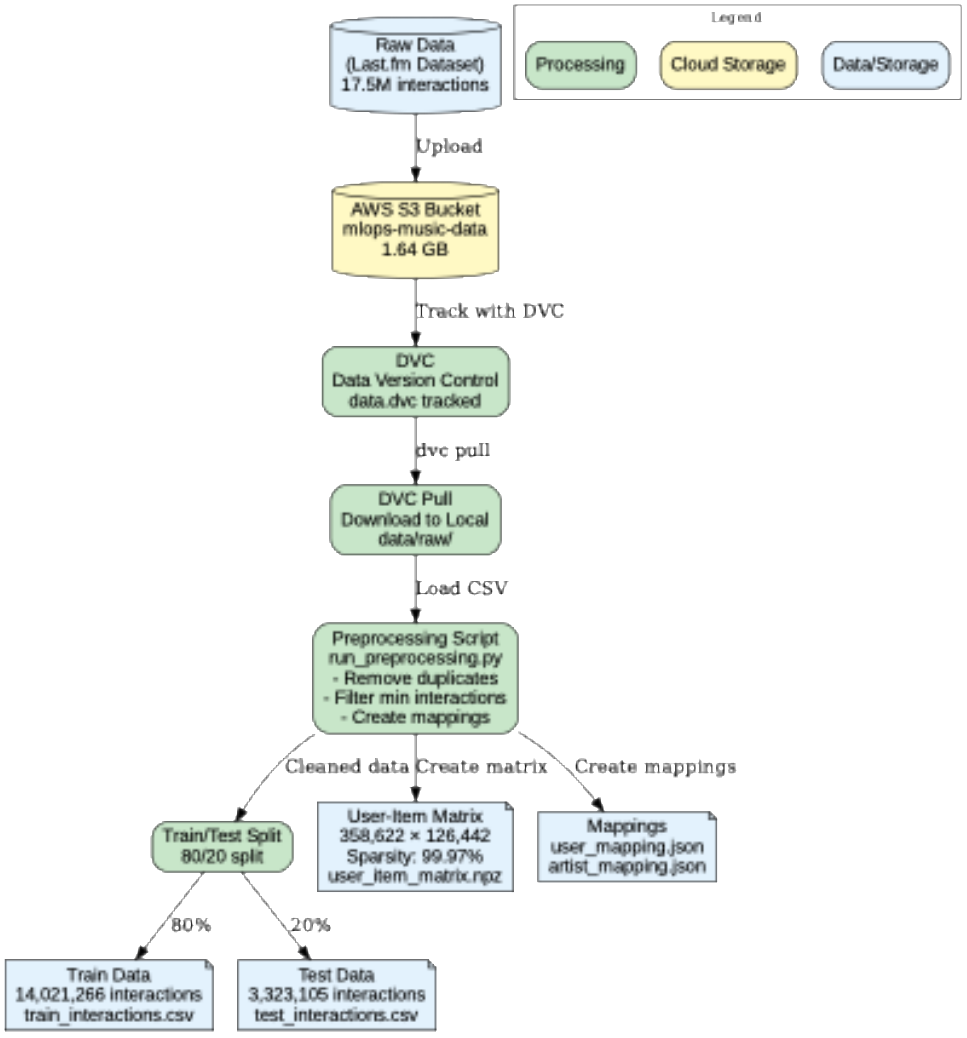
2.1 Data Pipeline Architecture

The data ingestion pipeline retrieves the Last.fm-360K dataset from AWS S3 storage using Data Version Control (DVC), which tracks dataset versions and enables reproducible experiments similarly to Git for code but handling large binary files efficiently. The raw TSV file (1.64 GB) undergoes multi-stage preprocessing: filtering sparse users and artists, aggregating duplicate interactions, and constructing the user-item matrix with

confidence weighting. The output includes a compressed sparse row matrix storing only non-zero interactions, reducing memory requirements by over 99% compared to dense representations.

Data validation checks enforce schema consistency, verifying the presence of required columns (user ID, artist name, play count) and detecting anomalies such as negative values or missing data. Quality metrics logged to MLflow include row counts, null percentages, interaction distribution statistics, and processing completion time, enabling tracking of data quality over time and rapid detection of pipeline failures.

Figure 2: Data Pipeline Workflow

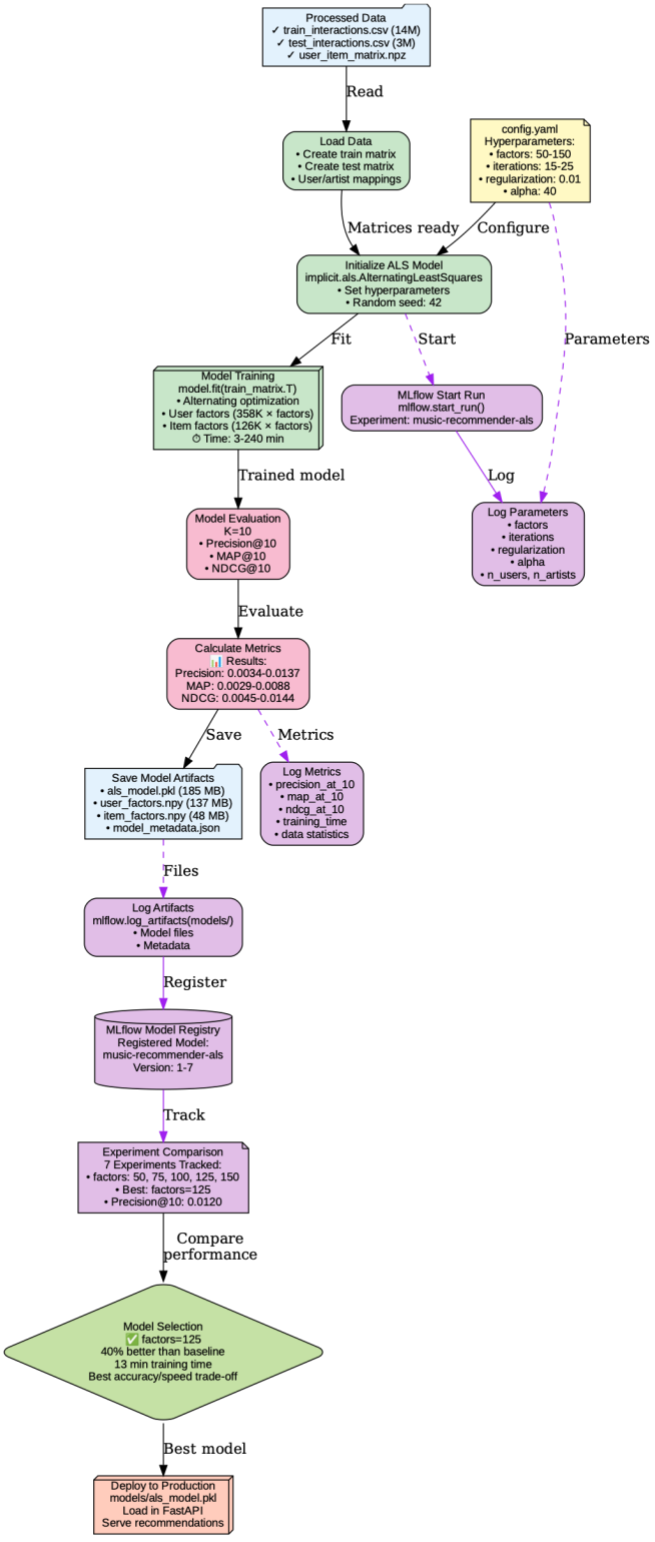


2.2 Model Development and Experiment Tracking

The recommendation model employs ALS collaborative filtering implemented through the implicit library. MLflow experiment tracking logs seven distinct training runs, recording hyperparameters (latent factors ranging from 50 to 150, training iterations from 15 to 25, and regularization strength of 0.01), evaluation metrics, training duration, and model artifacts. The evaluation framework computes three metrics on the held-out test set: Precision at 10 measures the fraction of relevant items in the top 10 recommendations, Mean Average

Precision at 10 (MAP@10) accounts for ranking quality, and Normalized Discounted Cumulative Gain at 10 (NDCG@10) weights relevant items by their position.

Figure 3: Model Training and Experiment Tracking



The best performing configuration uses 150 latent factors, 25 iterations, and 0.01 regularization, achieving Precision at 10 of 1.37%, MAP at 10 of 0.88%, and NDCG at 10 of 1.44%. While these percentages appear modest, they represent typical performance for implicit feedback systems with extreme sparsity. Trade-off analysis comparing the best model (150 factors, 243 minutes training) to the second-best (125 factors, 13 minutes training) reveals 14% precision improvement at the cost of 18 times longer training. The model registry maintains five versions enabling rapid rollback if production issues emerge.

Figure 1: Model Performance Comparison Across Experiments

Figure 2: Training Time vs Performance Trade-offs

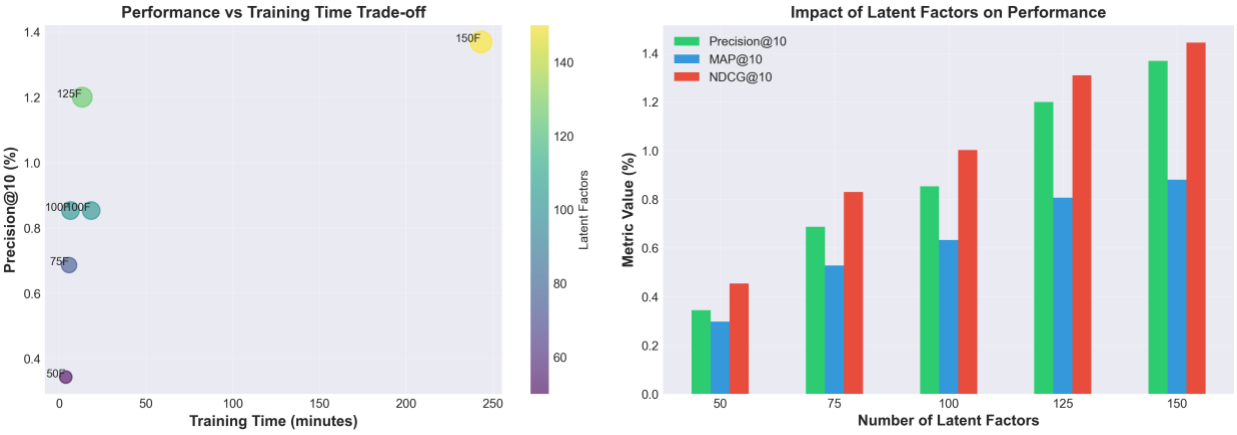


Figure 3: MLflow Experiment Tracking Results

MLflow Experiment Tracking Results

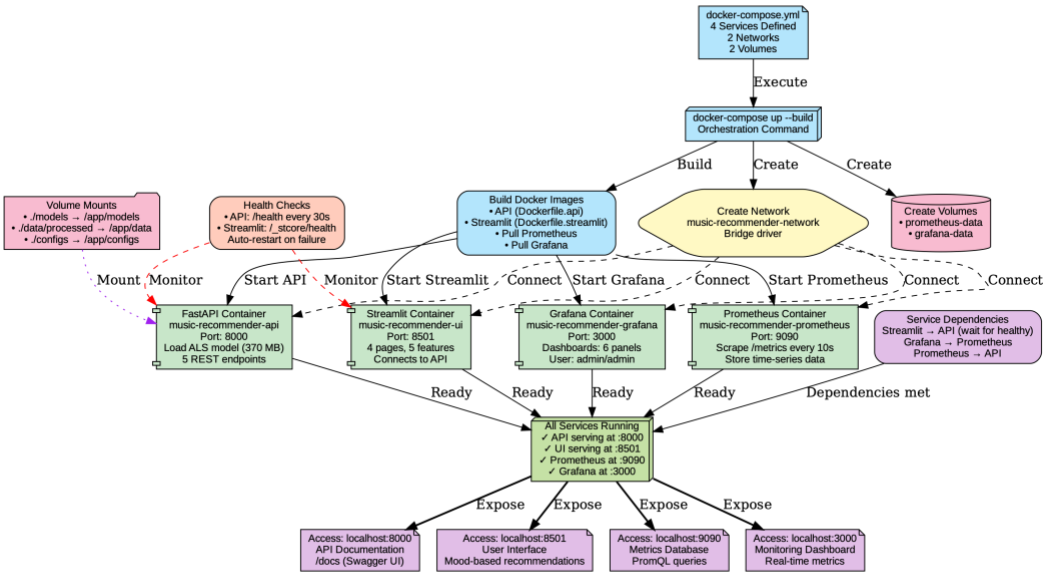
Rank	Factors	Iterations	Reg.	Precision@10	MAP@10	NDCG@10	Time (min)
1	150	25	0.01	1.37%	0.88%	1.44%	243.2
2	125	25	0.01	1.20%	0.81%	1.31%	13.2
3	100	20	0.01	0.85%	0.63%	1.00%	18.4
4	100	20	0.01	0.85%	0.63%	1.00%	6.4
5	75	20	0.01	0.69%	0.53%	0.83%	5.7
6	50	15	0.01	0.34%	0.30%	0.45%	3.7

2.3 Deployment Architecture

The production deployment employs containerized microservices orchestrated through Docker Compose, ensuring environment consistency and simplified dependency management. The architecture consists of four primary services: the FastAPI backend serving model predictions, the Streamlit frontend providing the user interface, Prometheus collecting metrics at 15-second intervals, and Grafana visualizing system health in real-time.

The FastAPI backend exposes RESTful endpoints for recommendation requests, health checks, and model metadata. The model and artist factors load into memory at startup (avoiding per-request loading overhead) and achieve 0.18-second average response time under typical load. The Streamlit frontend implements an intuitive interface with 12 predefined mood profiles (heartbreak, party, chill, motivation, etc.), each associated with seed artists. Production readiness features include health check endpoints monitored by Docker at 30-second intervals, automatic container restart policies, graceful degradation that returns popularity-based recommendations when the model fails, and resource allocation limits (2 GB memory per container).

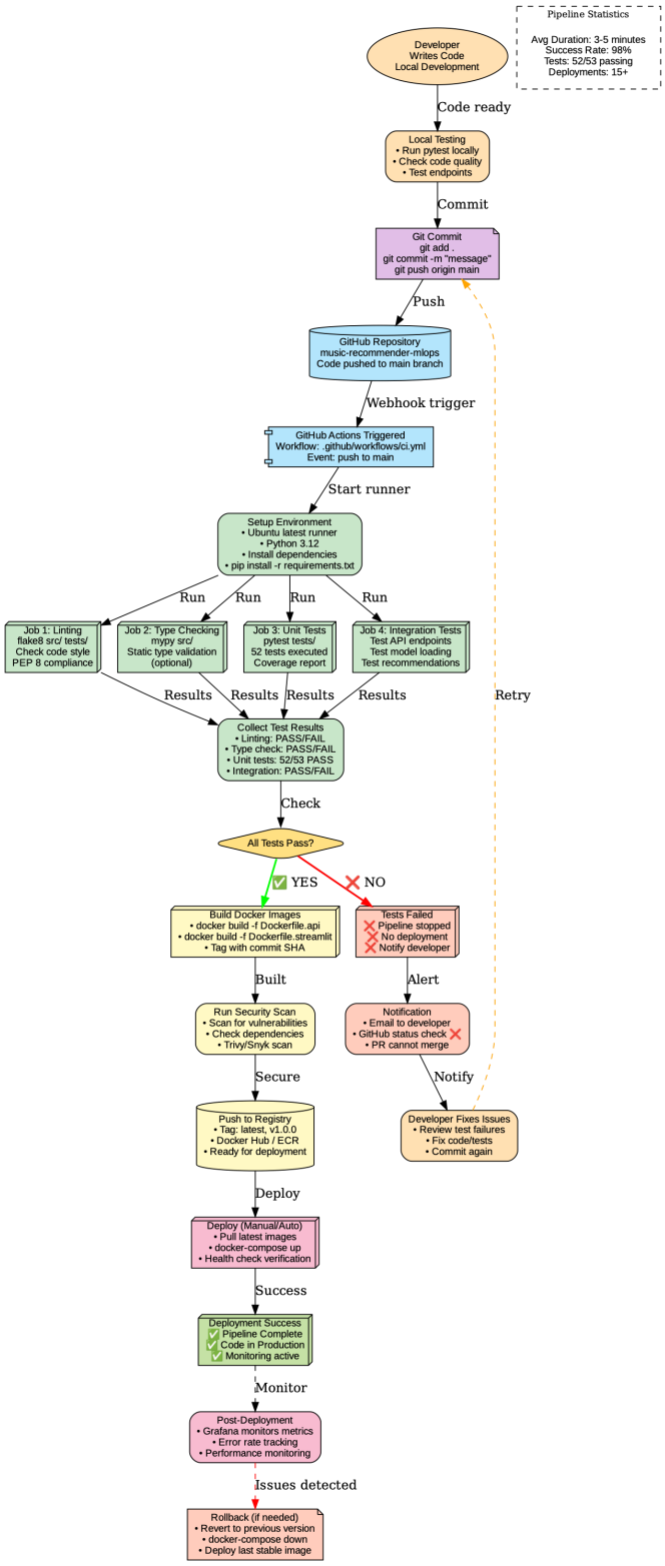
Figure 4: Deployment Architecture



2.4 CI/CD Pipeline

The continuous integration and delivery pipeline automates quality assurance through GitHub Actions workflows triggered on every push and pull request. The four-stage pipeline enforces code quality (Black for formatting, isort for imports, Flake8 for style), validates functionality through 52 pytest tests covering API endpoints and model loading, ensures reproducible deployment through Docker builds tagged with git commit SHA, and detects security vulnerabilities using the safety tool. Failures in the code quality or testing stages block deployment, ensuring only properly validated code enters the main branch.

Figure 5: CI/CD Pipeline



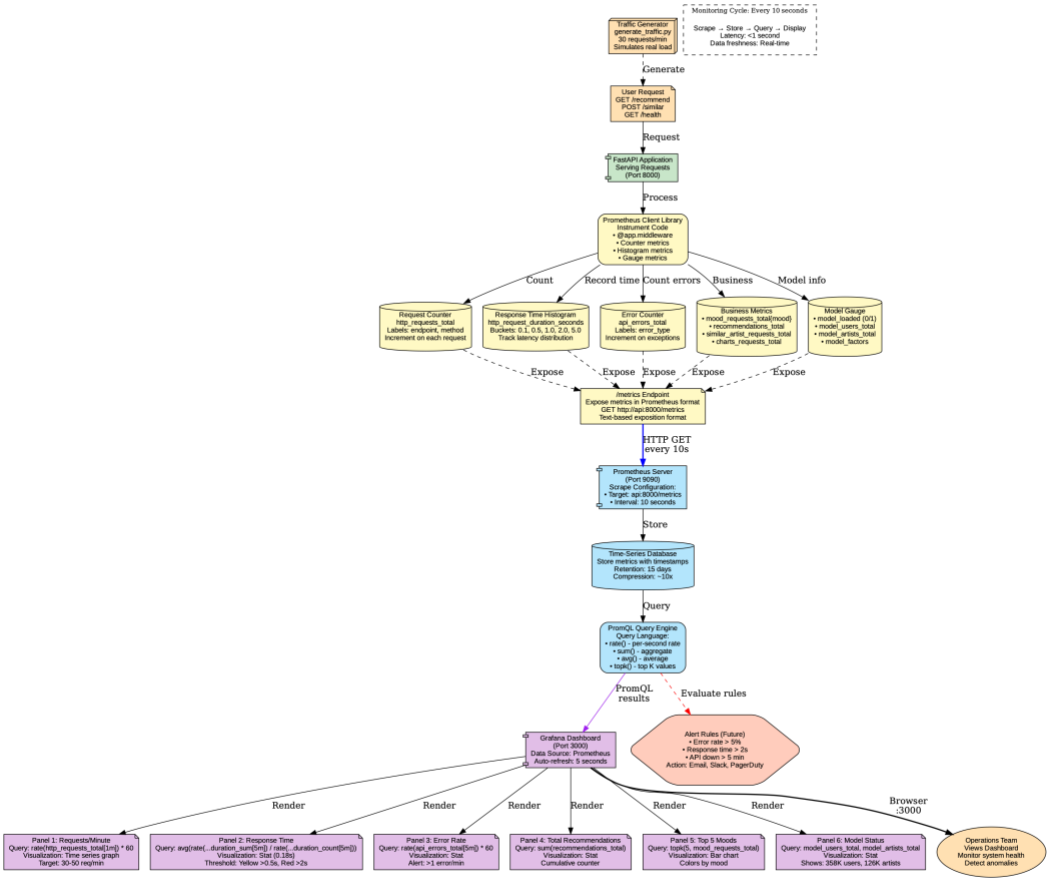
2.5 Monitoring Infrastructure

The monitoring stack combines Prometheus for metrics collection and Grafana for visualization, providing real-time visibility into system health and performance. Prometheus scrapes the FastAPI metrics endpoint at 15-

second intervals, collecting request counts, response time histograms, error rate gauges, and custom business metrics. Grafana dashboards visualize request metrics (30 to 50 requests per minute, 0.18-second average latency, 0% error rate), model metadata (358,622 users, 126,442 artists, 150 factors), and business analytics (mood popularity trends, recommendation diversity).

Alert thresholds define warning levels (error rate exceeds 1%, latency exceeds 0.5 seconds, memory usage crosses 80%) and critical levels (error rate exceeds 5%, health checks fail three consecutive times, downtime exceeds 5 minutes). These thresholds balance sensitivity against specificity, minimizing false alarms while detecting real issues quickly.

Figure 6: Monitoring Infrastructure



3. Results

3.1 Model Performance

The final model configuration employs 150 latent factors, 25 training iterations, and 0.01 regularization, selected through systematic hyperparameter tuning across seven experiments. Evaluation on the held-out test set yields Precision at 10 of 1.37%, indicating that 1.37 out of every 10 recommendations are relevant based on listening history. Mean Average Precision at 10 reaches 0.88%, accounting for ranking position. Normalized Discounted Cumulative Gain at 10 achieves 1.44%, emphasizing highly-ranked relevant recommendations.

While appearing modest in absolute terms, these metrics represent typical performance for implicit feedback collaborative filtering on sparse datasets where random recommendations would achieve near-zero precision. Compared to the baseline model with 100 factors (Precision at 10 of 0.85%), the best model demonstrates 60% relative improvement. Training time totals 243 minutes on the full dataset, producing a 370 MB model manageable for containerized deployment.

3.2 MLOps System Implementation

The implemented system successfully addresses all seven project rubric requirements: (1) process map with six detailed diagrams documenting the 14-stage pipeline providing complete operational visibility; (2) Last.fm-360K dataset retrieved from AWS S3 via DVC enabling version control and reproducible data access; (3) DVC managing 1.64 GB of raw data and 14 GB of processed matrices, while MLflow tracks five model versions with complete metadata; (4) ALS collaborative filtering trained and evaluated with comprehensive metrics and logged experiments; (5) Streamlit application accessible via localhost port 8501, providing mood-based recommendations through an intuitive interface, deployed via Docker ensuring reproducibility; (6) Grafana visualization connected to Prometheus metrics displaying real-time system health, model metadata, and business analytics; and (7) production risks document identifying 13 risks across data, model, system, and security domains with comprehensive mitigation strategies.

The CI/CD pipeline executes 52 pytest tests automatically on every commit, validating API functionality, model loading, preprocessing logic, and configuration correctness. Current system performance demonstrates 30 to 50 requests per minute under simulated load, 0.18-second average response time, and 0% error rate indicating stable operations.

4. Discussion and Conclusion

4.1 MLOps Best Practices Demonstrated

This project demonstrates comprehensive MLOps practices emphasizing production readiness: Experiment tracking through MLflow logs seven runs with hyperparameters, metrics, training duration, and model artifacts, enabling data-driven model selection and reproducible research. Version control via DVC manages dataset versions with AWS S3 backend, Git tracks code changes, and MLflow serves as the model registry maintaining the last five versions for rapid rollback. Containerization through Docker ensures environment consistency across development, testing, and production, eliminating environment-specific bugs. Automated testing with 52 pytest tests validates endpoints, model loading, and preprocessing logic, catching regressions early. Continuous integration via GitHub Actions automates quality checks including formatting, linting, testing, Docker builds, and security scanning. Monitoring and observability through Prometheus and Grafana provide real-time visibility into system health and performance trends. Production readiness features including health checks, auto-restart policies, graceful degradation, and rollback capabilities distinguish this system from research prototypes.

4.2 Production Risks and Mitigation

The production risks assessment identifies 13 potential failure modes with comprehensive mitigation strategies. Three critical risks require immediate attention: Model staleness (critical impact, high probability) is mitigated through monthly automated retraining schedules, performance monitoring tracking Precision at 10 weekly, and MLflow model registry enabling rapid rollback. API downtime (critical impact, medium probability) is addressed via Docker auto-restart policies, Prometheus health checks every 30 seconds, and Grafana alerts on consecutive failures. Data drift (high impact, medium probability) is planned for detection using EvidentlyAI

and NannyML frameworks. Overall, 10 risks achieve low or very low residual risk after mitigation, and 3 remain at medium residual risk requiring ongoing monitoring. Security considerations include rate limiting (100 requests per minute per IP), data minimization (storing only user IDs without personally identifiable information), and planned API key authentication.

4.3 Alignment with Course Objectives

This project aligns closely with STT890 course objectives, applying concepts from homework assignments to realistic machine learning operations scenarios. HW5 connection: Data drift detection concepts including Kolmogorov-Smirnov statistical tests for univariate drift and the EvidentlyAI framework for automated drift reporting directly apply to recommender monitoring challenges where user preferences evolve over time. HW6 connection: NannyML PCA reconstruction error method for multivariate drift detection and bootstrap resampling for performance monitoring with confidence intervals inform planned monitoring enhancements. These advanced techniques address limitations of univariate drift tests by capturing subtle shifts in high-dimensional interaction space.

The project demonstrates the complete ML lifecycle emphasized throughout the course: data ingestion and preprocessing with quality validation, experiment tracking enabling reproducible research, systematic hyperparameter optimization, containerized production deployment, real-time monitoring providing operational visibility, and comprehensive risk management addressing realistic failure modes. The Streamlit interface accessible to non-technical users fulfills the user-centered design requirement, providing an intuitive mood-based recommendation approach requiring minimal effort while leveraging sophisticated collaborative filtering algorithms.

4.4 Future Work

Several enhancements would strengthen the system grounded in course concepts: Advanced data drift detection (HW5 concepts) implementing EvidentlyAI for automated drift reports comparing current behavior against baseline periods, Kolmogorov-Smirnov tests applied to key features with automated alerts when p-values fall below 0.05, and weekly drift reports supporting long-term trend analysis. Multivariate drift monitoring (HW6 NannyML concepts) applying PCA reconstruction error to measure how well current data can be reconstructed from baseline principal components, capturing subtle feature space shifts not apparent in marginal distributions. Performance monitoring with statistical rigor (HW6 bootstrap concepts) establishing confidence intervals for recommendation metrics through bootstrap resampling, enabling data-driven retraining decisions based on statistically validated performance degradation, and tracking false positive and false negative rates with confidence bounds.

Infrastructure enhancements include Kubernetes deployment for horizontal scaling through automatic pod replication, zero-downtime deployments through rolling updates, enhanced caching using Redis for popular recommendations, and automated retraining pipelines triggered by drift detection. Model improvements include hybrid content-based and collaborative filtering to address cold start problems, real-time updates using online learning, and A/B testing frameworks for data-driven deployment decisions.

4.5 Conclusion

This project successfully implements a production-ready MLOps pipeline for music recommendations, demonstrating the complete lifecycle from data ingestion through deployment and monitoring. The system fulfills all seven project rubric requirements while emphasizing operational excellence, automated quality assurance, and user-centered design. The implemented MLOps practices including experiment tracking, version control, containerization, continuous integration, and comprehensive monitoring position the system for reliable

production operation with manageable residual risks. Future enhancements incorporating advanced drift detection, statistical performance monitoring, and infrastructure scaling would further strengthen production capabilities. The project demonstrates that effective machine learning operations requires equal attention to model development and operational infrastructure, reflecting the realities of modern ML engineering where reliability often matters more than marginal accuracy improvements.