MLOps Lab1 Assignment

Task Summary

In this assignment, you are expected to conceptualize and design a machine learning project on which you will be working during the course. Your submission should cover the high-level project architecture, data requirements, services decomposition, metrics and specific requirements for each component of the system.

Important: you will be implementing the system during the course so it shouldn't be too complicated. It should include at least 1 ML model. You should be able to get or gather data for the system.

Deliverables:

- Define the problem statement and objectives.
- Diagram of the overall system architecture (data flows, components, and interactions).
- Description of the ML models involved, including short description.
- Specify the datasets and data updates.
- Describe(high level) the data preprocessing steps and how data will be managed and stored.
- Break down the system into microservices.
- Describe the role and functionality of each service in the context of the overall system.
- Explain the communication between services (e.g., REST APIs, message queues).
- List the operational requirements for each service, like: scalability, performance.
- Specify the tools and technologies that you'll use for the implementation.
- Define how the performance of the machine learning model will be measured.
- Outline the criteria for success of the overall system from both a technical and business perspective

Design doc guidelines:
https://github.com/eugeneyan/ml-design-docs

1. Overview

This document outlines the design and implementation of an automated MLOps system for CrossEncoder reranker models within a Retrieval-Augmented Generation (RAG) application. The system provides end-to-end machine learning lifecycle management including automated data drift detection, model retraining, evaluation, and deployment. The solution leverages AWS

SageMaker for scalable ML pipelines, Airflow for orchestration, MLflow for experiment tracking, and FastAPI for model serving, all deployed on AWS EC2 infrastructure using Docker containers.

2. Motivation

RAG applications rely heavily on the quality of document retrieval and ranking to provide accurate responses. The reranker component is critical for improving retrieval precision by reordering documents based on query-document relevance scores. However, reranker models can degrade over time due to:

- **Data drift**: Changes in query patterns, document types, or user behavior
- **Performance degradation**: Model accuracy decreasing as new data patterns emerge
- **Manual maintenance overhead**: Time-consuming manual model updates and deployments

The current manual approach to model updates is inefficient and reactive. An automated MLOps system is essential to maintain optimal reranker performance, reduce operational overhead, and ensure consistent service quality in production environments.

3. Success Metrics

Business Goals

- Increase NDCG@10 scores by 15% through automated model optimization
- Decrease manual model maintenance effort by 80%
- Reduce time from data drift detection to model deployment from weeks to hours
- Maintain consistent search quality with automated performance monitoring

Technical Metrics

- NDCG@10 > 0.85, MRR > 0.75, MAP@10 > 0.70
- 99.9% uptime for FastAPI serving endpoint
- Successful automated retraining cycles within 4-hour SLA
- Drift detection running on 7-day rolling windows

4. Requirements & Constraints

4.1 Functional Requirements

- Monitor data distribution changes with configurable thresholds via automated drift detection

- Compare new models against production baselines before deployment via champion-challenger framework
- Zero-downtime deployment
- Comprehensive logging of training runs, hyperparameters, and metrics
- Model versioning
- Performance monitoring (preferable real-time)

4.2 Non-Functional Requirements

- Model inference < 100ms for single query-document pair scoring
- Handle 1000+ requests per second during peak traffic
- Support model training on datasets up to 1M query-document pairs
- Monthly AWS operational costs < $500 for development environment
- Secure API access with proper authentication and data encryption
- Containerized services with clear separation of concerns

4.3 Constraints

- Budget is limited to AWS free tier and low-cost instances for initial deployment
- Compliance with data retention policies and user privacy requirements
- Single-region deployment on AWS with eventual multi-region consideration
- Focus on CrossEncoder architectures compatible with sentence-transformers library

4.4 What's in-scope & out-of-scope?

In-Scope:

- CrossEncoder reranker model training and deployment
- Data drift detection and automated retraining
- MLflow-based experiment tracking and model registry
- FastAPI model serving endpoint
- Airflow workflow orchestration
- Champion-challenger model evaluation

Out-of-Scope:

- Initial document retrieval system optimization
- Embedding model training or fine-tuning
- Multi-model ensemble approaches
- Real-time feature engineering

- Advanced A/B testing frameworks

5. Methodology

5.1 Problem Statement

The reranker optimization is framed as a supervised learning problem where we predict relevance scores for query-document pairs. The system implements a binary classification approach with relevance labels (0/1) while utilizing ranking metrics (NDCG, MRR, MAP) for evaluation. The problem is further enhanced with a data drift detection component that triggers automated retraining when distribution shifts exceed predefined thresholds.

5.2 Data

Training Data Sources:
Supabase database: primary source containing query-document pairs with relevance labels
- Schema: `{query: str, document: str, relevance: int (0/1), source: str, created_at: timestamp}`
- Volume: ~10K labeled pairs with continuous growth
- Quality: human-annotated relevance judgments

External Data Integration:

- Qdrant vector database: document embeddings and similarity scores for feature enhancement
- OpenAI API: query processing and document analysis for additional features

Serving Data:

- **Input**: query-document pairs via FastAPI endpoint
- **Output**: relevance scores (0-1 probability) for ranking

5.3 Techniques

Model Architecture:

- **Base model**: CrossEncoder using `cross-encoder/ms-marco-MiniLM-L-6-v2`
- **Fine-tuning**: Sentence-transformers CrossEncoderTrainer with CachedMultipleNegativesRankingLoss
- **Training strategy**: Supervised learning with contrastive loss and negative sampling

Data Processing:

- Remove duplicates, filter by text length, validate relevance labels
- Target 30% positive samples using undersampling techniques
- Text length statistics, word counts for drift detection
- Tokenization and encoding handled by transformers library

Drift Detection Methods:

- **Statistical tests**: Kolmogorov-Smirnov test, Population Stability Index (PSI)
- **Text similarity**: TF-IDF cosine similarity between time periods
- **Label distribution**: Chi-square test and KL divergence for relevance shift detection

5.4 Experimentation & Validation

Offline Evaluation:

- Main metrics are NDCG@10, MRR@10, MAP@10, Hits@10, Precision/Recall
  - Primary: NDCG@10 > current production model
  - Guardrail: no degradation in MRR@10 or MAP@10 below -0.05 threshold
- Validation strategy is 85/15 train-test split with time-based splits to prevent data leakage
- Champion-challenger framework has to be implemented with NDCG@10 improvement > 0.01 threshold and automatic reversion if performance degrades post-deployment

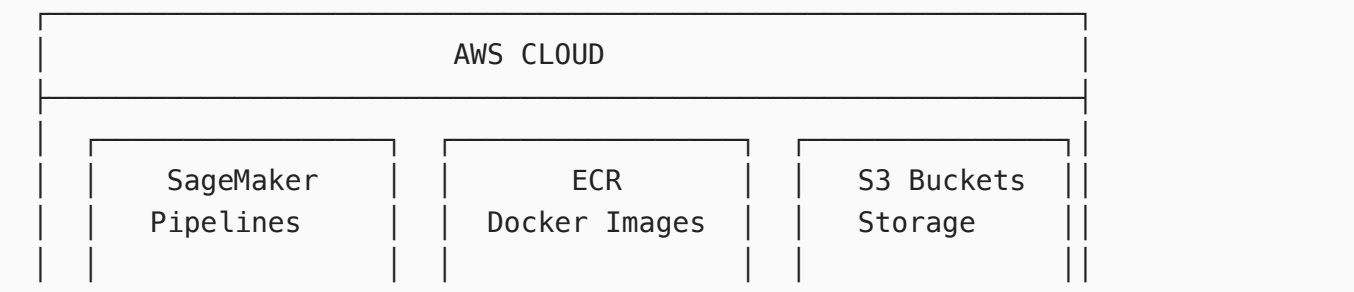
5.5 Human-in-the-loop

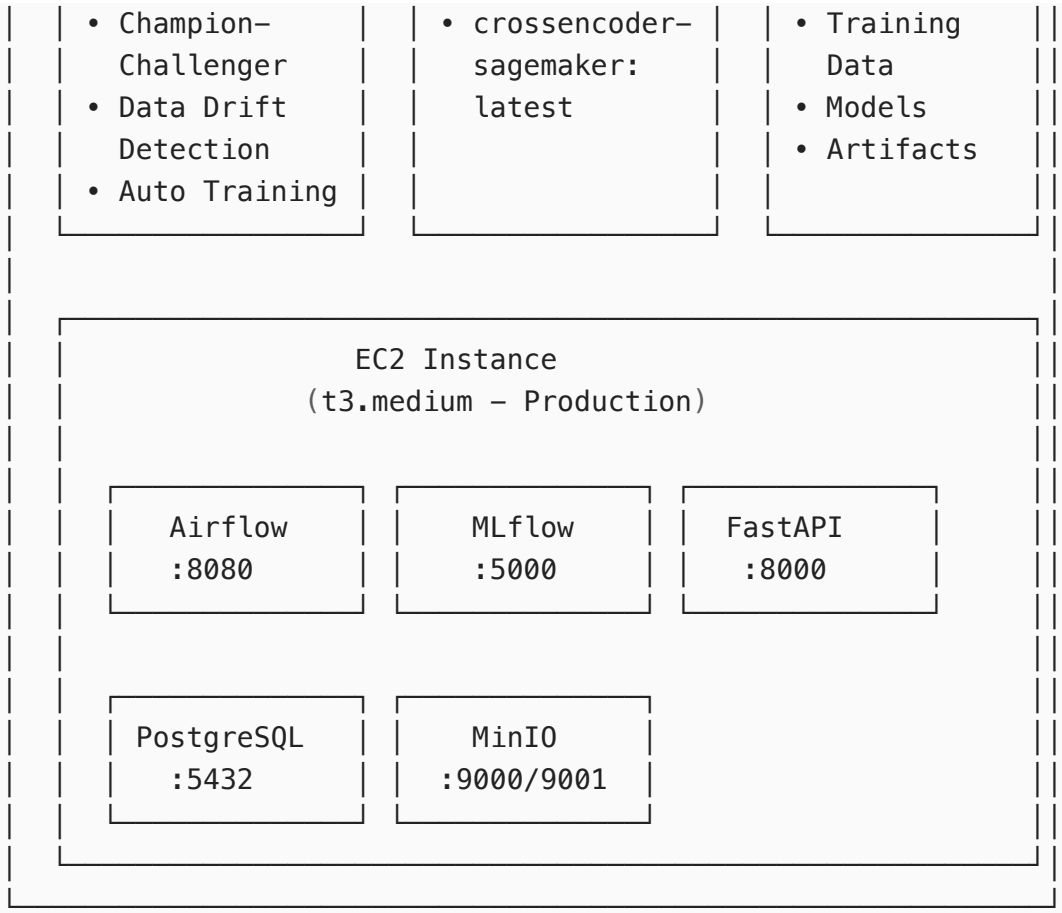
The best expected future production strategy is quality assurance with:

- Weekly manual review of model predictions on sample queries
- User feedback collection through Streamlit interface
- Subject matter expert review of model updates before deployment
- Manual deployment controls for critical updates

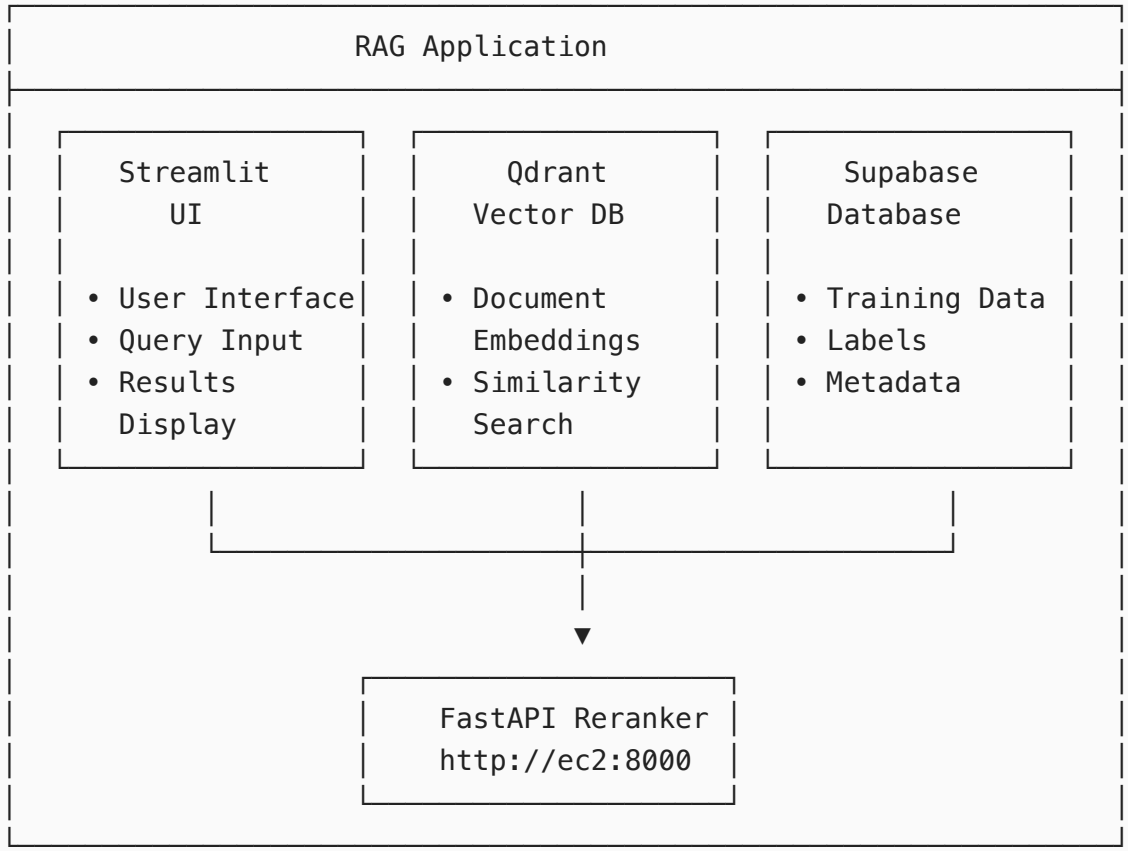
6. Implementation

6.1 High-level Design





External Application Layer:



6.2 Infrastructure

The system deploys on AWS infrastructure in the eu-north-1 region using a containerized architecture. The core compute platform is an EC2 t3.medium instance (2 vCPU, 4GB RAM) running Docker containers orchestrated through Docker Compose for simplified multi-service deployment. Storage architecture combines EBS volumes for persistent data with S3 buckets for artifacts, while networking utilizes VPC configurations with security groups and Application Load Balancer for production traffic distribution.

Core Container Services:

- **Airflow**: workflow orchestration (webserver + scheduler)
- **MLflow**: experiment tracking and model registry
- **FastAPI**: model serving endpoint
- **PostgreSQL**: metadata storage for Airflow and MLflow
- **MinIO**: S3-compatible object storage for artifacts

6.3 Performance (Throughput, Latency)

The system targets specific performance benchmarks to ensure production readiness. Model inference latency must remain below 100ms per query-document pair for single requests, while batch processing should achieve sub-50ms per pair latency for batches of 10 or more items. End-to-end API response times including network overhead are targeted to stay under 200ms.

Throughput capacity should support over 1000 requests per second on the t3.medium instance configuration. The scaling strategy employs horizontal scaling with multiple FastAPI containers deployed behind a load balancer to handle increased traffic demands, with in-memory model caching achieving sub-millisecond prediction times once models are loaded.

6.4 Security

API security relies on JWT token-based authentication for FastAPI endpoints, while internal service communication benefits from Docker network isolation. AWS access follows the principle of least privilege through carefully configured IAM roles and policies, with sensitive data management utilizing environment variables and AWS Secrets Manager.

Network security measures:

- **Security groups**: firewall configurations restricting access to necessary ports only
- **HTTPS termination**: SSL/TLS encryption at load balancer level
- **VPC architecture**: private subnets for services, public subnet for load balancer

6.5 Data Privacy

Data privacy compliance encompasses comprehensive measures to protect user information and meet regulatory requirements. The system implements automated 90-day retention policies for personal data in training datasets, with query preprocessing including anonymization steps to remove personally identifiable information. Access controls enforce role-based permissions combined with comprehensive audit logging, while encryption protocols secure data both at rest and in transit.

The perspective is to implement GDPR compliance features: API endpoints for data removal requests (Right to Deletion), export functionality for user information, and integration with consent management platforms.

6.6 Monitoring & Alarms

Application metrics tracking includes request latency, error rates, and throughput monitoring through Prometheus and Grafana integration. Infrastructure monitoring utilizes CloudWatch for resource tracking, while model performance monitoring tracks key metrics like NDCG and MRR with automated alerts for performance degradation.

- **Critical Alerts**: service downtime, high error rates (PagerDuty integration)
- **Warning Alerts**: performance degradation, resource utilization (Slack notifications)
- **Model Alerts**: srift detection events, failed training runs

6.7 Cost

AWS infrastructure:

- EC2 t3.medium (24/7) ~$60/month
- EBS storage (100GB) ~$10/month
- S3 storage (50GB) ~$1/month
- Data transfer ~$5/month
- **Subtotal ~$76/month**

SageMaker usage:

- Pipeline execution (4 runs) ~$10/month
- Training instances (ml.m5.large) ~$30/month
- **Subtotal ~$50/month**

External services:

- Supabase Pro ~$25/month

- Qdrant Cloud ~$20/month
- **Subtotal ~$45/month**

Total estimated cost ~$171/month

Cost Optimization:

- Spot instances for training workloads (50% savings)
- Reserved instances for production (30% savings)
- Auto-scaling for variable workloads

6.8 Integration Points

- Supabase database → RESTful API for training data retrieval
- Qdrant database → gRPC API for embedding and similarity data
- OpenAI API → HTTP API for query/document processing
- Streamlit → HTTP calls to FastAPI reranker endpoint
- Airflow ↔ SageMaker → AWS SDK for pipeline trigger and monitoring
- MLflow ↔ FastAPI → Model registry API for production model loading
- All Services ↔ PostgreSQL → Database connections for metadata storage

6.9 Risks & Uncertainties

Technical Risks:

- Model performance regression through champion-challenger framework
- Risk of unnecessary retraining, addressed with tunable thresholds
- Single point of failure on EC2, planning multi-AZ deployment

Business Risks:

- Data quality degradation impact on model training, addressed with data validation pipelines
- SageMaker costs scaling with usage, implementing cost monitoring and budgets
- Data privacy regulations, maintaining legal review process

Operational Uncertainties:

- Unknown growth in traffic, planning auto-scaling implementation
- Future requirements for ensemble methods or transformer models
- Potential challenges with external API changes or rate limits

7. Appendix

7.1 Alternatives

1. **Alternative 1: Kubernetes-based Deployment**
   - **Pros**: Better scalability, industry standard, resource efficiency
   - **Cons**: Higher complexity, longer setup time, additional operational overhead
   - **Decision**: Rejected for initial version due to complexity; considered for future iterations
2. **Alternative 2: Serverless Architecture (Lambda + SageMaker)**
   - **Pros**: Cost efficiency for variable workloads, automatic scaling, reduced maintenance
   - **Cons**: Cold start latency, complexity in persistent connections, vendor lock-in
   - **Decision**: Rejected due to latency requirements and complexity of ML workflows
3. **Alternative 3: On-premise Deployment**
   - **Pros**: Full control, no cloud costs, data privacy
   - **Cons**: Hardware procurement, maintenance overhead, limited scalability
   - **Decision**: Rejected due to operational complexity and scaling limitations

7.2 Experiment Results

Baseline model performance:

- NDCG@10: 0.823 ± 0.012
- MRR@10: 0.756 ± 0.018
- MAP@10: 0.689 ± 0.015
- Training Time: 45 minutes on ml.m5.large

Drift detection sensitivity analysis:

- Threshold 0.15: 15% false positive rate, 95% true positive rate
- Threshold 0.25: 5% false positive rate, 88% true positive rate
- Threshold 0.35: 1% false positive rate, 72% true positive rate
- Selected: 0.25 for optimal balance

7.3 Performance Benchmarks

Inference latency (t3.medium instance):

- Single pair: 45ms ± 5ms
- Batch size 10: 38ms ± 3ms per pair
- Batch size 50: 35ms ± 2ms per pair

Training performance:

- 10K samples: 25 minutes on ml.m5.large
- 50K samples: 1.5 hours on ml.m5.large

7.4 Milestones & Timeline

1. **Infrastructure setup (Weeks 1-2)**
   - EC2 instance provisioning and Docker setup
   - Basic service deployment (Airflow, MLflow, FastAPI)
   - Security group and networking configuration
2. **ML pipeline development (Weeks 3-5)**
   - SageMaker pipeline implementation
   - Data drift detection algorithms
   - Model training and evaluation scripts
3. **Integration & testing (Weeks 6-7)**
   - End-to-end pipeline testing
   - Performance optimization
   - Monitoring and alerting setup
4. **Production deployment (Week 8)**
   - Production environment deployment
   - Documentation and handover
   - Initial production validation

7.5 Glossary

- Champion-Challenger — model deployment strategy comparing new models against production baseline
- CrossEncoder — transformer architecture that processes query-document pairs jointly
- Data Drift — statistical change in input data distribution over time
- NDCG — Normalized Discounted Cumulative Gain, ranking metric for information retrieval
- RAG — Retrieval-Augmented Generation, AI architecture combining retrieval and generation
- Reranker — model that reorders retrieved documents by relevance to improve search quality

7.6 References

1. Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks
2. Nogueira, R., & Cho, K. (2019). Passage Re-ranking with BERT

3. MLOps Principles: https://ml-ops.org/
4. AWS SageMaker Pipelines: https://docs.aws.amazon.com/sagemaker/latest/dg/pipelines.html
5. Eugene Yan's ML Design Docs: https://eugeneyan.com/writing/ml-design-docs/
6. CrossEncoder Documentation: https://www.sbert.net/examples/applications/cross-encoder/README.html