SaaS Product Intelligence Platform

Interview Explanation & Project Context

■ What is this project?

This is a **production ML system** that answers 'Why did [metric] change?' using internal company data. It's NOT a generic chatbot or simple RAG system. It's a sophisticated platform that combines modern ML techniques (dense retrieval, sparse search, learning-to-rank) with constrained reasoning to provide accurate, evidence-based answers.

Real Example:
User: 'Why did user activation drop in March?'
System: 'The onboarding redesign in Release 2.3 caused a 20% activation drop due to UI changes in the signup flow. (Confidence: 87%, Sources: Release notes, Support tickets)'

■ Problem it Solves

Product managers, data analysts, and executives need quick answers to metric changes. Manual investigation is slow and error-prone. Naive RAG systems hallucinate (make up answers). This system provides accurate, cited, production-grade intelligence.

■■ How It Works (5-Step Pipeline)

Step 1: Retrieval (Hybrid)

Given a query, we search documents two ways: (1) Dense - convert query to 384-dim embedding, find similar docs via FAISS, (2) Sparse - BM25 keyword search. Merge results by combined score.

Step 2: Ranking (Learning-to-Rank)

Score retrieval candidates with 6 features: dense_score, sparse_score, doc_length, term_overlap, recency_decay, feedback_signal. LightGBM LambdaRank model reorders by usefulness. Optimizes NDCG (ranking quality metric).

Step 3: LLM Reasoning (Constrained)

Top-3 ranked docs are fed to LLM which synthesizes an answer. Three critical constraints: (1) Citations - every claim must cite a source, (2) Confidence - computed as 40% doc_count + 40% rank_score + 20% answer_length, (3) Refusal - refuses if confidence < 0.5.

Step 4: Monitoring

Record latency, recall, NDCG, refusal rate, confidence. Compare against baselines. Detect drift (e.g., latency > 1000ms). Alert if system degradation detected.

Step 5: Feedback Loop

Log every interaction (query, answer, citations, confidence) to JSONL. User provides feedback (helpful/not helpful). Feedback generates training signals for ranking model retraining.

■ Why This Approach?

Why Hybrid Retrieval?

Dense search alone misses keyword matches. Sparse search alone misses semantic meaning. Combining both maximizes recall - we retrieve candidates that match semantically OR by keywords. Ranking then picks the most useful.

Why Learning-to-Rank (LambdaRank)?

Simple relevance scoring (BM25 + dense) doesn't account for usefulness. LambdaRank optimizes for ranking order - it learns from feedback which docs users find helpful. This is superior to classification (binary relevant/irrelevant) because ranking order matters.

Why Constrained LLM (not free-form)?

LLMs hallucinate - they make up plausible-sounding but false answers. Constraints prevent this: (1) Citations anchor answers to source documents, (2) Confidence prevents overconfident wrong answers, (3) Refusal admits uncertainty. This is production-grade AI.

Why Monitoring & Feedback Loops?

ML systems degrade in production. Monitoring catches problems (latency creep, recall drop). Feedback loops enable continuous improvement - user feedback trains better ranking models. This is how Netflix/Google/Meta do it.

■ Key Technical Decisions

FastAPI (not Django/Flask): Async support, auto-generated API docs, native async/await, excellent performance

SentenceTransformers all-MiniLM-L6-v2: 384-dim embeddings, great semantic quality, fast, open-source, no API keys

FAISS (not Pinecone/Weaviate): Self-hosted, open-source, production-proven, no vendor lock-in

LightGBM LambdaRank (not neural): Interpretable, fast, proven for ranking, graceful degradation

JSONL Logging (not DB): Append-only, immutable audit trail, compatible with training pipelines

Feedback-driven retraining: Enables continuous improvement without manual labeling

Drift detection: Catches production issues automatically

■ Common Interview Questions

Q: Why hybrid retrieval instead of just dense or sparse?

A: Dense captures semantic meaning (e.g., 'activation drop' finds 'user engagement decline'). Sparse captures exact keywords (e.g., 'March' or 'Release 2.3'). Together, they have higher recall. Ranking then picks the best.

Q: How does LambdaRank differ from simple ranking?

A: LambdaRank optimizes for ranking order using pairwise comparisons. It learns that if doc1 is more helpful than doc2, it should be ranked higher. This is better than binary classification because ranking order matters in search.

Q: How do you prevent hallucination?

A: Three layers: (1) Constrained prompting - LLM must cite sources, (2) Confidence scoring - weighted by citation count and rank score, (3) Refusal - if confidence < 0.5, system says 'I don't know'. We'd rather admit uncertainty than be confidently wrong.

Q: How do you measure quality?

A: NDCG (Normalized Discounted Cumulative Gain) for ranking quality. Recall for retrieval coverage. Latency for performance. Refusal rate and confidence for safety. User feedback (helpful %) for real-world impact. All tracked with baselines for drift detection.

Q: How does the system improve over time?

A: User feedback is logged to JSONL. We generate NDCG labels from feedback: label = helpful_binary × confidence_score × (not_refused). Periodically retrain the LambdaRank model on accumulated feedback. Versioned models enable rollback if needed.

Q: What would you do in production?

A: (1) Deploy on Kubernetes, (2) Use Prometheus for metrics, (3) Set up alerts for drift, (4) Run weekly feedback analysis, (5) Retrain models monthly, (6) Monitor latency P95, recall, NDCG, refusal rate, (7) Log all interactions for debugging and improvement.

■ What You Should Know to Discuss This

Information Retrieval: Dense vs sparse search, FAISS, BM25, hybrid retrieval, recall vs precision

Learning-to-Rank: Pairwise ranking, LambdaRank, NDCG metric, how feedback drives ranking improvement

Constrained LLM: How to constrain LLMs (citations, confidence), preventing hallucination, refusal logic

ML Monitoring: Drift detection, baseline thresholds, metrics tracking, alert systems

Feedback Loops: How user feedback generates training data, label generation, continuous retraining

API Design: FastAPI, Pydantic, REST endpoints, async/await, request/response validation

Data Validation: Schema validation, drift detection, data quality checks

Testing: Unit tests, integration tests, smoke tests, pytest

Deployment: Docker, CI/CD, GitHub Actions, containerization

Production Thinking: Error handling, graceful degradation, logging, monitoring, rollback strategies

■ Elevator Pitch (30 seconds)

I built a production ML system that answers 'Why did [metric] change?' for SaaS companies. It combines hybrid retrieval (dense + sparse search), learning-to-rank (LambdaRank on user feedback), and constrained LLM reasoning (citations + confidence + refusal) to prevent hallucination. The system monitors itself, learns from feedback, and continuously improves. It's deployed with Docker, tested with pytest, and has a CI/CD pipeline.

■ Key Files to Reference

API & Pipeline: app/api.py (5 endpoints), app/pipeline.py (orchestration)
Retrieval: app/retrieval/dense_retrieval.py, sparse_retrieval.py, hybrid_retrieval.py
Ranking: app/ranking/ranker.py (LambdaRank), features.py (6-dim features)
LLM: app/llm/constrained.py (citations + confidence + refusal)
Monitoring: app/monitoring.py (metrics, drift detection)
Feedback: app/feedback.py (JSONL logging, stats)
Training: training/train_ranker.py (feedback-driven retraining)
Tests: tests/test_smoke.py (4/4 passing), test_retrieval.py, test_ranking.py, test_llm.py, test_pipeline.py

Document generated: February 17, 2026