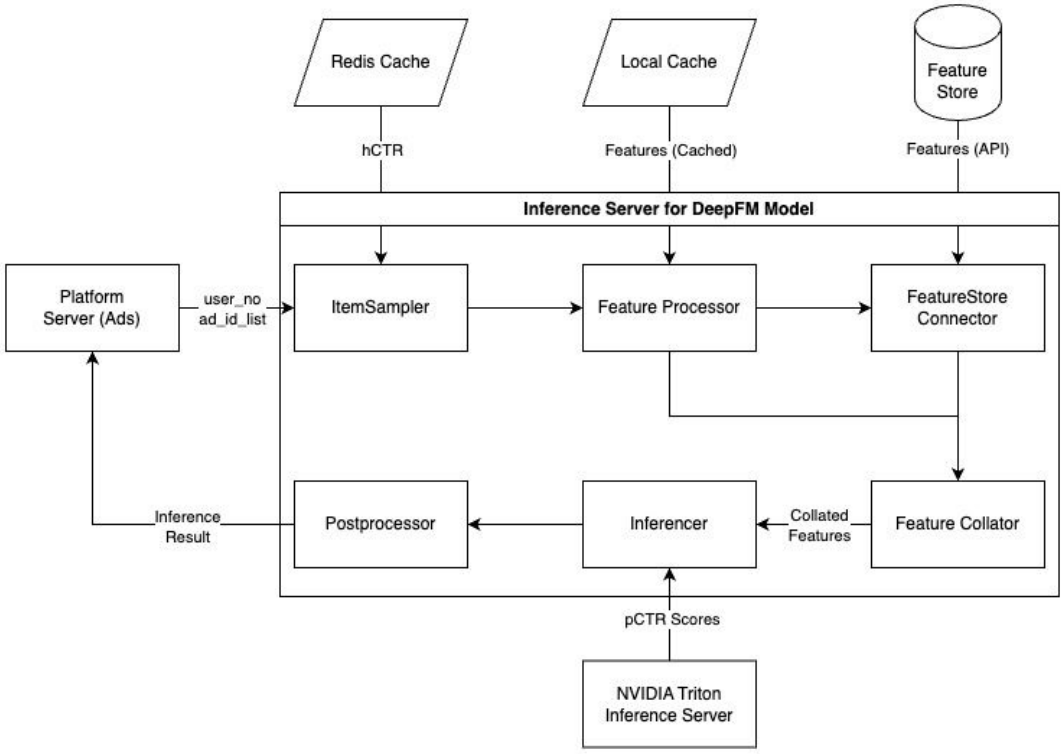
In Keun Kim

SOFTWARE ENGINEERING PORTFOLIO

ik2619@columbia.edu • (332) 265-6478 • linkedin.com/in/nearkim • github.com/nearKim

This portfolio supplements my resume by demonstrating experience in designing and delivering production-grade AI systems, robust backend platforms, and large-scale data infrastructures.

[1] DeepFM Personalization Inference Cluster



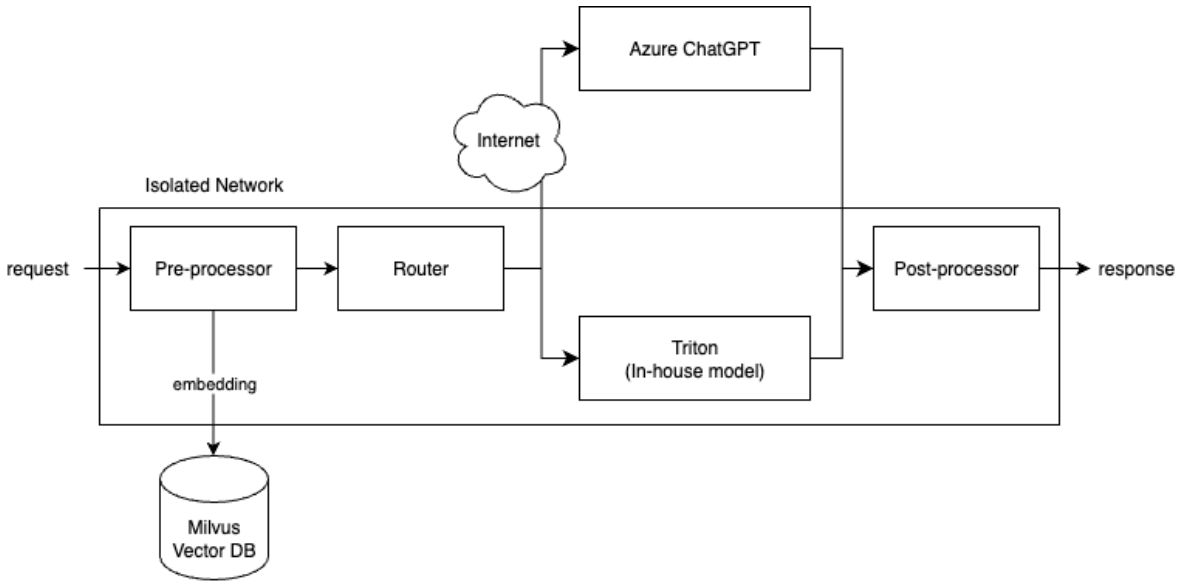
Summary: I built a production inference service (DeepFM on ONNX) to predict pCTR for personalized ad inventories, integrating MLflow for model management and Aerospike for real-time features. The architecture supports one control model and eight variants for continuous A/B experimentation with hourly model refreshes.

Key Results:

- **6,000+ RPS** sustained with **p95 ≤70ms** latency.
- Model release time **6h → 30m** via dynamic model loading and zero-downtime deploys.
- Automatic model detection and on-demand updates with multi-variant routing.

Tech: Kotlin (Spring Boot/WebFlux), ONNX, MLflow, Aerospike, Kafka, Triton, Redis, Sentry, Micrometer

[2] Unified LLM Server Platform (In-house + Azure OpenAI)



Summary: I created a unified platform that abstracts in-house LLMs and Azure OpenAI behind a single API, standardizes request/response payloads, adds intelligent batching, and integrates Milvus for semantic retrieval to power RAG-style applications.

Key Results:

- **One standardized interface** reduced integration time for partner teams.
- Implemented accurate semantic search, providing the foundation required for Retrieval-Augmented Generation (RAG) applications.
- Deployed in an automated UI/UX evaluation tool and piloted in a virtual customer center.

Tech: Kotlin (Spring Boot/WebFlux), Azure OpenAI, Milvus, Triton

[3] Refactoring of Inference Services Applying Domain-Driven Design and JVM Profiling

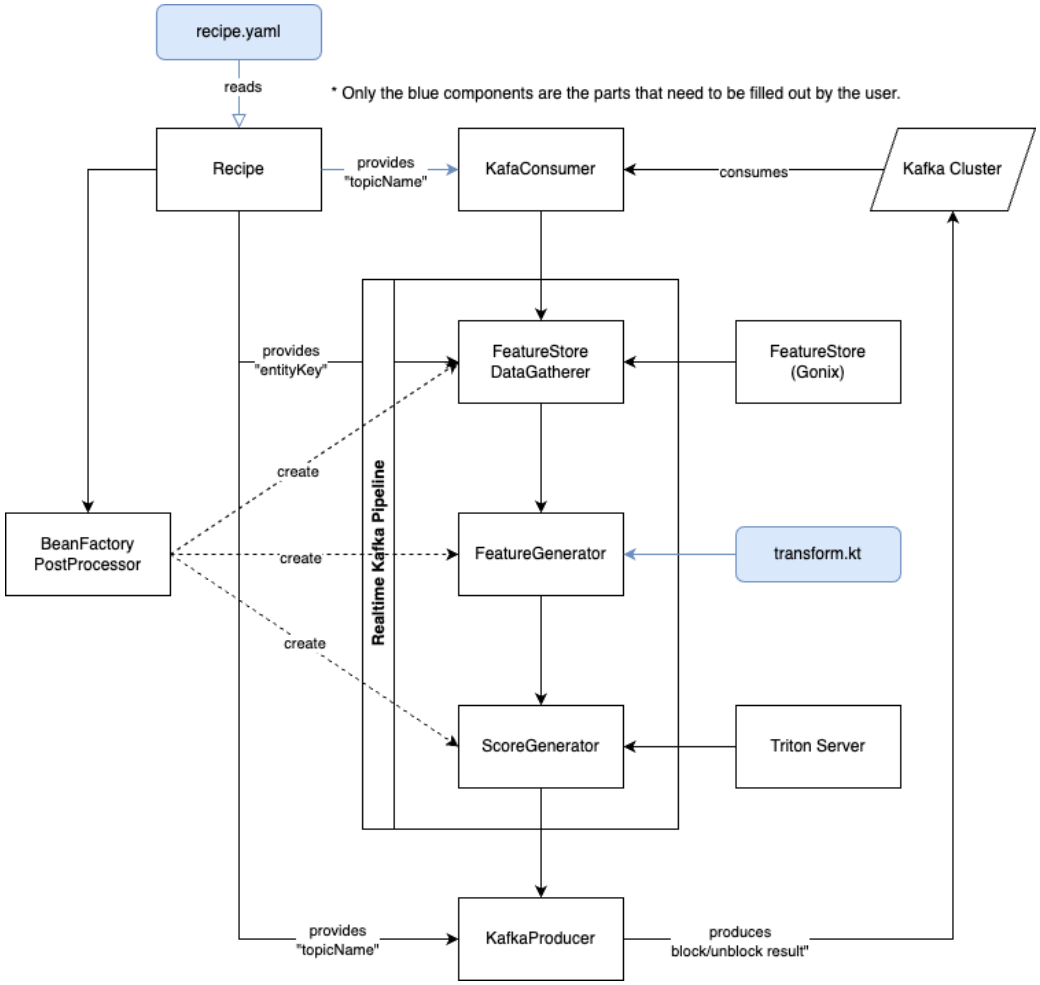
Summary: I led the revival of a neglected inference server project that teammates had avoided due to technical debt. By restructuring the monolithic codebase into Domain, Application, Presentation, and Infrastructure layers (DDD) and applying JVM-level profiling, I transformed the codebase into a maintainable and collaborative system.

Key Results:

- **25%** reduction in Kubernetes pod memory **(8GB → 6GB)**.
- **1,500+ duplicate LOC removed**, eliminating a primary source of bugs and the reason the team avoided touching the service.
- Endpoint delivery time reduced from **1 day → 3 hours**.
- Restored the project from “abandoned” status to an actively developed, multi-contributor service.

Tech: Kotlin (Spring Boot/WebFlux), DDD, async-profiler.

[4] YAML-Based No-Code Framework for ML Pipelines



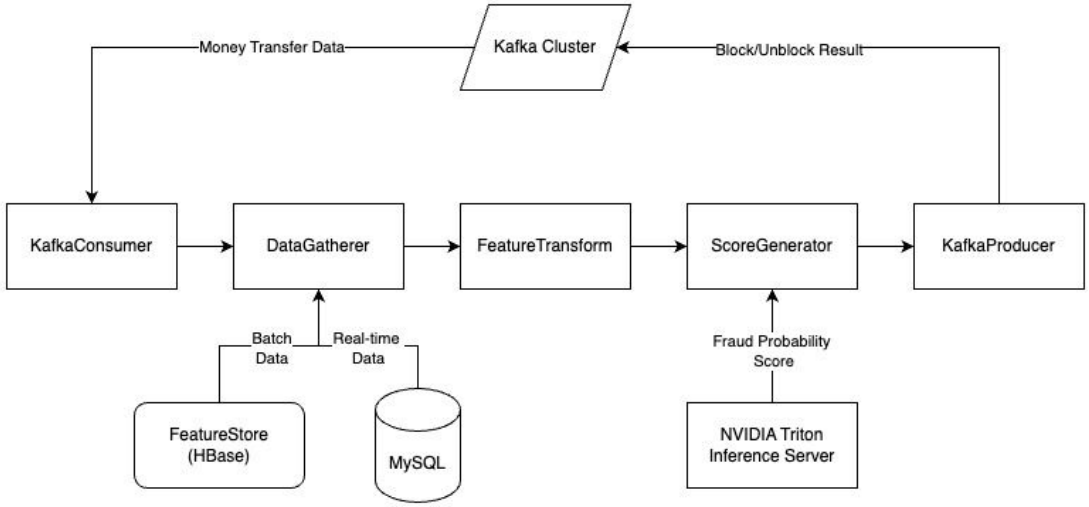
Summary: I engineered a declarative framework that turns YAML into Spring Beans to provision Kafka consumers and related resources—enabling nontechnical PMs/analysts to launch and update ML pipelines without writing code.

Key Results:

- **Zero additional engineering time** to create or update real-time pipelines.
- **Instant service creation** on commit with standardized pipeline logic.
- Faster iteration for anomaly and fraud-detection use cases.

Tech: Kotlin (Spring Boot/WebFlux), Kafka, CatBoost, Triton, HBase, Apache Phoenix, MySQL

[5] Real-Time Fraud Detection System (FDS)



Summary: I built an end-to-end FDS using CatBoost to score risk and automatically block/unblock activity in real time, closing gaps that rule-based systems miss. Two dedicated Kotlin Kafka services handle scoring and actions.

Key Results:

- Maintained **<30% false-positive rate** while automating high-risk blocks/unblocks.
- Processed **>1TB** of real-time features across MySQL/HBase.
- Consistent **p95 200ms / p999 600ms** latency SLAs.

Tech: Kotlin (Spring Boot), Kafka, CatBoost, Triton, HBase, MySQL.

[6] JuiceFS Object Storage for Facial Images (Kubernetes)

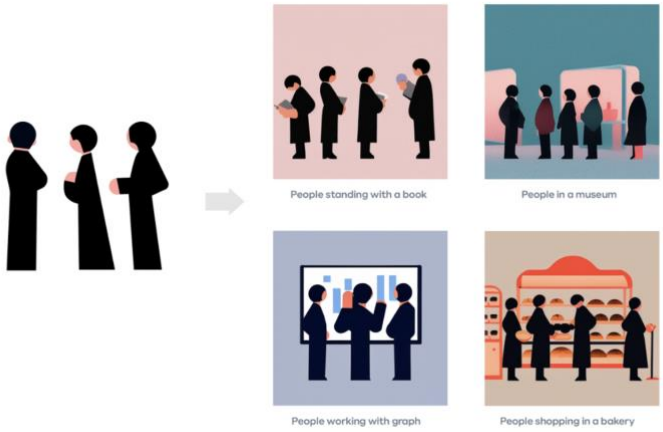
Summary: I deployed a POSIX-compliant object store using JuiceFS for over 20 million facial images, integrated as a persistent volume for the Airflow Kubernetes Operator, and made it compatible with HDFS/S3 backends.

Key Results:

- Delivered a **distributed file system for 20M+ images** with seamless data access.
- Eliminated the dedicated NAS cluster and associated operational overhead.
- Integrated cleanly into DS workflows with **zero reported issues**.

Tech: Python, JuiceFS, Redis, Kubernetes, Airflow.

[7] Automated Image Generation Service Backend (Tosst)



Summary: I developed Tosst, a web service that generates design-compliant image resources aligned with Toss's branding guidelines. Leveraging a Data Scientist-built model deployed via stable-diffusion-web-ui, the service processes user prompts, generates images, stores outputs in S3, handles errors, operates in event-driven design, and logs metadata in MySQL for traceability.

Key Results:

- Eliminated outsourcing by enabling in-house automated image generation.
- Saved ≈ ₩100M (~USD 72.5K) per month in external design costs.

Tech: Python (FastAPI, SQLAlchemy, asyncio), Stable Diffusion, stable-diffusion-web-ui , MySQL

References:

- https://toss.tech/article/ai-graphic-generator-1
- https://toss.tech/article/ai-graphic-generator-2