Building a Recommendation Engine at ESPN Without Slowing Delivery

AI as Your Team's Multiplier

Katerina Iliakopoulou-Zanos

Recommendations Lead / Principal ML Engineer, ESPN

Many teams are using AI.

Not every team is getting faster.

The gap no one talks about

> *"Teams adopt AI tools while architecture decisions don’t change"*

> *"AI generates code fast, but ‘it runs’ doesn’t mean ‘it’s right’"*

> *"More code gets written, yet delivery doesn’t actually speed up"*

This is a story about how we rebuilt ESPN's recommendation platform in eight months.

By leveraging AI as the multiplier.

Sports recommendations are different.

Live events

Content relevance changes by the minute during games

Strong fandoms

Users have deep, identity-level preferences

Editorial voice

Domain rules and editorial curation matter — heuristics go far

Heuristics were both the strength and the ceiling.

The system we inherited

Signals

- User embedding (heuristic)
- Explicit favorites
- Implicit favorites
- Popularity / recency
- Trendingness
- Editorial boosts

Retrieval + ranking = one coupled step

Where it broke down

Impression skew

Most impressions went to a small subset of inventory. Long-tail content was starved.

Universal rules

Boosts and filters applied to all users equally. We missed what each user actually wanted.

Coupled pipeline

Retrieval and ranking tangled in one query. Improving one meant touching everything.

Where AI didn't multiply.

The Two-Tower project

Team used AI coding assistants to build a Two-Tower embedding model over 3 months. The code ran. It looked like progress.

But underneath:

~80% of the code was unnecessary.
Feature generation coupled into training.
No architectural plan. Just a goal.

The antipattern

AI wasn't prompted with architectural constraints — just a goal. It built a plausible system that moved fast in the wrong direction.

We paused the project.

AI without a strategy
doesn't slow you down.
It speeds you up in the wrong direction.

So what does AI as a multiplier actually look like?

AI as onboarding co-pilot

Codebase exploration

Navigated and understood an unfamiliar codebase fast

Data exploration

SQL queries, understanding schemas, mapping telemetry flows

Tool onboarding

Databricks was new to the entire team — AI as always-available expert

AI compressed the "learn the landscape" phase — the part that usually takes weeks.

First shipped change: Thompson Sampling

The fix

Principled exploration that surfaces long-tail content in a controlled way without blowing up the user experience.

Addressed the impression skew directly.

Result

Meaningful lift in watch minutes

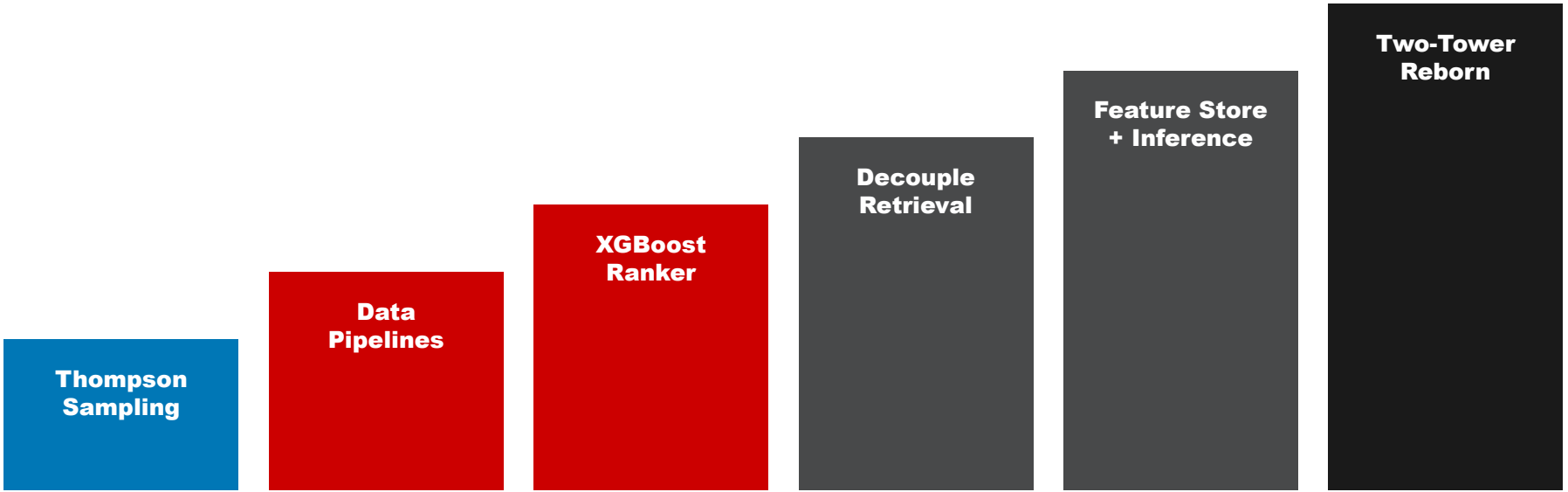
Clear problem → scoped solution → AI-assisted prototyping and evaluation → shipped.

Once we had the right strategy, AI became the engine.

Every piece had to be independently valuable.

Ship value now. Unlock the next step.

Incremental modernization



Each step shipped value on its own — and unlocked the next.

Data Foundations

Medallion-architecture pipelines

What we built

Raw telemetry → clean datasets → features + training data. The unsexy infrastructure that makes everything else possible.

2 ML Engineers + 1 Data Engineer

Fully productionalized in 2 months

This is normally a separate team's job.

How AI augmented this work

Pipeline code generation: pattern-heavy, clear strategy and rules on data schemas.

Onboarding to Databricks: a tool nobody on the team had used. AI as always-available expert.

SQL queries and data exploration: understanding unfamiliar schemas at speed.

AI let ML engineers operate across a wider surface area.

XGBoost Ranker

First ML ranking model → replaced the heuristic

Why XGBoost, not a neural ranker?

Deliberately chose a simpler model. Could deploy within the existing reco API. Didn't need the inference server yet.

Ship ranking impact now. Invest in infrastructure next.

Same 2 ML Engineers

0 → production in 3 months

Including feature development through the new pipelines.

AI built it fast. Choosing XGBoost over a neural ranker at this stage was human judgment.

Decoupling + shared infrastructure

Two-stage architecture

Broke the monolithic ES query into independent retrieval sources feeding a ranking layer.

Improve each stage independently.

Feature Store + Inference Server

2 ML Engineers. Built and tested in prod in 2 months.

Unlocked: sophisticated features + deep neural network models in production.

How AI removed blockers across the board

BLOCKER		WITH AI
✗ Most of the team didn't know Databricks	→	✓ AI made it a non-issue
✗ DevOps work slowing down ML engineers	→	✓ Resolved with AI assistance
✗ Code reviews on unfamiliar codebases	→	✓ AI as onboarding co-pilot
✗ Evaluating model performance offline	→	✓ Tooling built with AI assistance in days
✗ Understanding messy telemetry data	→	✓ SQL exploration at 10x speed

Remember the model we paused?

Two-Tower Model: Reborn

- Stripped the unnecessary 80%
- Removed coupled feature generation
- Replatformed on the Feature Store + Inference Server

Same model. Same AI tools.

This time, AI was multiplying the right strategy.

8 months. ~5 engineers. No delivery pauses.

Thompson Sampling

Live — meaningful lift

Data Pipelines

Productionalized

XGBoost Ranker

In production

Two-Stage Architecture

Retrieval + ranking decoupled

Feature Store + Inference

Operational

Two-Tower Model

Replatformed on real foundations

Where AI is the multiplier. And where it's not

AI compressed the work

- Code generation for well-defined patterns
- Onboarding to unfamiliar tools & codebases
- Code reviews across team boundaries
- SQL + data exploration at speed
- Offline evaluation tooling
- DevOps tasks that block non-specialists

Human judgment was irreplaceable

- What to build and in what order
- Spotting structural flaws in AI output
- Scoping problems before solving them
- Metric selection and evaluation design
- The decision to pause and build foundations
- Knowing when "it runs" ≠ "it's right"

How to start thinking about AI on your team

AI is a multiplier of your engineering strategy.

Feed it a clear plan, and your team ships in months what used to take years.

Feed it a vague goal, and you build the wrong thing faster than ever.

---

1. **Audit how your team uses AI right now**
   Is it multiplying a clear plan or generating code without one?

2. **Find your Thompson Sampling**
   What's the smallest principled change you can ship this month that proves the direction?

3. **Name the foundation you're skipping**
   What's the unsexy infrastructure your team keeps deferring? That's probably your bottleneck.

Thank you.

Katerina Iliakopoulou-Zanos

Recommendations Lead / Principal ML Engineer, ESPN

linkedin.com/in/katerinazanos