AITss • 2026

From Prototype to Production

Building Real-World AI Systems

A practical guide to shipping AI that works at scale

THE PROBLEM

Cool Demo ≠ Real Product

Every AI demo looks magical. Shipping it is a different story.

✨ The Demo

Perfect inputs
Ideal conditions
Manual QA
No real users

🔥 Production

Edge cases everywhere
Unpredictable outputs
Real latency & cost
Anger from real users

🎯 The Gap

Error handling
Observability
Cost management
Reliability patterns

This talk is about crossing that gap.

AGENDA

What We'll Cover

01

The AI App Stack

LLMs, RAG, APIs: how modern AI apps are structured

02

The Hard Problems

Latency, cost, hallucinations, and unpredictability

03

Production Patterns

Caching, fallbacks, evals, and guardrails

04

Observability & Ops

How to know what's actually happening

05

Lessons Learned

War stories and practical takeaways

SECTION 01 · ARCHITECTURE

The Modern AI App Stack

USER / CLIENT	Web, Mobile, API consumers
ORCHESTRATION	LangChain, custom agents, prompt chaining
LLM APIs	OpenAI, Anthropic, Gemini, Mistral, local
RAG / MEMORY	Vector DBs (Pinecone, pgvector), retrieval
TOOLS & APIS	Functions, web search, code execution
INFRA / OPS	Logging, eval pipelines, cost tracking

ARCHITECTURE · RAG

Retrieval-Augmented Generation (RAG)

The #1 pattern for grounding LLMs in real knowledge



When to use RAG

Private/recent data not in training set. Document Q&A, customer support, knowledge bases.

Chunking matters

Chunk too small = lost context. Chunk too large = noisy retrieval. 512 - 1024 tokens with overlap is a good start.

Retrieval quality

Your LLM is only as good as what you retrieve. Invest in re-ranking, hybrid search (dense + sparse).

Common pitfalls

Fetching irrelevant chunks, ignoring metadata filters, not evaluating retrieval separately from generation.

SECTION 02 · HARD PROBLEMS

What Actually Goes Wrong

⏱️ Latency

LLM calls are slow. p99 can be 10 - 30s. Users expect < 1s. Streaming helps perception but not reality.

💸 Cost

Token costs add up fast. A naive chatbot at scale can burn $10k/month before you notice.

🎲 Non-Determinism

Same prompt, different output every time. Temperature=0 helps but doesn't eliminate it.

🌀 Hallucinations

Models confidently produce false information. Citations don't mean correctness.

📉 Model Drift

Providers update models silently. Your evals pass on v1, fail on v1.1.

🔐 Prompt Injection

Users (or docs) can hijack your system prompt. A real attack vector, not just a theory.

HARD PROBLEMS · LATENCY

Taming Latency

~3s
Avg LLM first token

30s+
Worst-case p99

Streaming

Stream tokens as they arrive. Even if total time is 8s, users see words after 300ms. Perception is everything.

Prompt Caching

Cache repeated prompt prefixes (system prompts, RAG docs). Anthropic and OpenAI both support this, with up to 90% cost and latency reduction.

Parallel Calls

Don't chain LLM calls sequentially when they're independent. Run them in parallel with Promise.all().

Model Selection

Use small fast models (Haiku, GPT-4o-mini) for routing, classification, and simple tasks. Save large models for reasoning.

Edge Caching

Cache deterministic responses (FAQ, static content) at the CDN/edge layer. Some requests should never hit the LLM.

Timeout Strategy

Always set timeouts. Fail fast with a fallback rather than making users wait forever. 30s max, then degrade gracefully.

HARD PROBLEMS · COST

Cost: The Invisible Bomb

Token costs are small per call. They're catastrophic at scale if you're not watching.

Potential Cost Reduction by Strategy



💡 Set token budgets, alert on anomalies, and log cost per request from day one.

SECTION 03 · PATTERNS

Production Patterns That Work

1 LLM Router

Route requests to cheap/fast models for simple tasks, premium models only for complex ones. Can cut costs by 60–80%.

2 Structured Outputs

Use JSON mode or tool calling to force structured responses. Never parse free-text LLM output with regex.

3 Fallback Chain

Primary model → cheaper model → cached response → static fallback. Always have a graceful degradation path.

4 Guardrails Layer

Input validation + output validation. Run a fast classifier on outputs before serving them to users.

5 Eval-Driven Dev

Build your eval suite before optimizing. If you can't measure quality, you can't improve it safely.

6 Human-in-the-Loop

For high-stakes actions, require human approval. AI drafts, human confirms. Don't automate irreversible actions.

PATTERNS · PROMPTING

Prompt Engineering at Scale

Prompts are code. Treat them that way.

✅ DO

Version-control your prompts

Use system prompts for persona & constraints

Include few-shot examples for consistent format

Specify output format explicitly (JSON schema)

Test prompt changes with your eval suite

Separate concerns: routing vs reasoning vs formatting

❌ DON'T

Don't rely on long, brittle mega-prompts

Don't concatenate strings to build prompts

Don't skip prompt regression tests

Don't put sensitive data in prompts (PII!)

Don't assume the model follows instructions 100%

Don't over-engineer before you have real data

PATTERNS · EVALS

You Can't Ship What You Can't Measure

Functional Evals

Does the output satisfy the user intent?
Unit tests for your LLM. Use assert + LLM-as-judge.

Safety Evals

Does it refuse when it should? Does it comply when it should?
Adversarial testing matters.

Quality Evals

Tone, length, format, factuality.
Compare output pairs. Human preference data is gold.

Regression Evals

Run on every prompt/model change.
If your score drops, you broke something.

💡 LLM-as-judge: use GPT-4 or Claude to score outputs automatically. Cheap, scalable, good enough for iteration.

SECTION 04 · OBSERVABILITY

If It's Not Logged, It Didn't Happen

🔍 Traces

Full request lifecycle: prompt in → tokens out → latency → cost. Langfuse, Helicone, or custom.

📊 Metrics

Token usage, cost/request, error rate, latency p50/p95/p99, fallback rate.

📝 Logs

Structured logs with user ID, session, model, prompt hash. Enables debugging and auditing.

🔔 Alerts

Cost spike alerts, error rate thresholds, latency regressions. Page on call at 3am, not the user.

Tools: Langfuse • Helicone • Datadog LLM Observability • Custom OpenTelemetry

ADVANCED · AGENTS

When to Use Agents (and When Not To)

Agents = LLMs that take actions. More power, more failure modes.

Good Use Cases

Multi-step research & synthesis

Code generation + execution

Form-filling & browser automation

Long-horizon tasks with checkpoints

Tool-use pipelines with clear success criteria

Danger Zones

Irreversible actions (send email, delete data)

Financial transactions without approval

Unbounded web browsing / tool use

Tasks without a clear done state

Customer-facing with no human fallback

Best Practices

Minimal tool permissions (least privilege)

Confirmation before destructive actions

Hard step limits to prevent infinite loops

Full audit trail of every action taken

Graceful abandonment when stuck

ADVANCED · SECURITY

Security & Trust in AI Systems

Prompt Injection

HIGH

Malicious content in user input or retrieved docs that overrides your system prompt. Treat all external content as untrusted.

Data Leakage

HIGH

PII, secrets, and confidential data in prompts gets logged, trained on, or exposed in outputs. Scrub before sending.

Jailbreaking

MED

Users crafting prompts to bypass your guardrails. Layered defenses: input filter + output filter + classifier.

Supply Chain

MED

LLM providers can change models, terms, or go offline. Avoid hard coupling to a single provider.

🔒 Principle: Never trust LLM output unconditionally. Validate, sanitize, audit.

SECTION 05 · LESSONS

War Stories & Lessons Learned

01

Start with the eval suite, not the feature

Without evals, every change is a gamble. Set up even 20 golden test cases before you optimize anything.

02

Latency is a product problem, not an infra problem

Users don't understand LLM latency. Streaming, skeletons, and progressive disclosure are your tools.

03

Your prompt is your product

A 20% prompt improvement often beats a model upgrade. Invest time here before spending on bigger models.

04

Cost surprises are almost always preventable

Log token usage per request from day 1. Set budgets and alerts before launch, not after the bill arrives.

05

Model updates will break you silently

Subscribe to provider changelogs. Pin model versions. Run evals on every deployment, not just locally.

FRAMEWORK · MATURITY

AI System Maturity Model

Where are you? Where do you need to be?

L1 Prototype	Hardcoded prompts | No evals | Direct API calls | No observability
L2 MVP	Basic error handling | Some logging | Cost awareness | Streaming
L3 Production	Eval suite | Fallbacks | Cost limits | LLM observability | Guardrails
L4 Scale	Model routing | Prompt caching | A/B testing | SLA monitoring | Evals in CI
L5 Optimized	Fine-tuned models | Custom eval frameworks | Automated regression | Full audit trail

TAKEAWAYS

The Short Version



Architecture first: understand the stack before writing a line of code.



Build evals before you build features. Measure everything.



Latency kills UX. Stream, cache, and right-size your models.



Cost creep is real. Log and alert from day one.



Guardrails aren't optional. Validate inputs, validate outputs.



Observability is your superpower. If it's not logged, it didn't happen.



Iterate fast, but regression-test every prompt change.

Build AI that works.

Not just demos.

Questions?

From Prototype to Production: Building Real-World AI Systems

fahimbinamin.com LinkedIn X GitHub