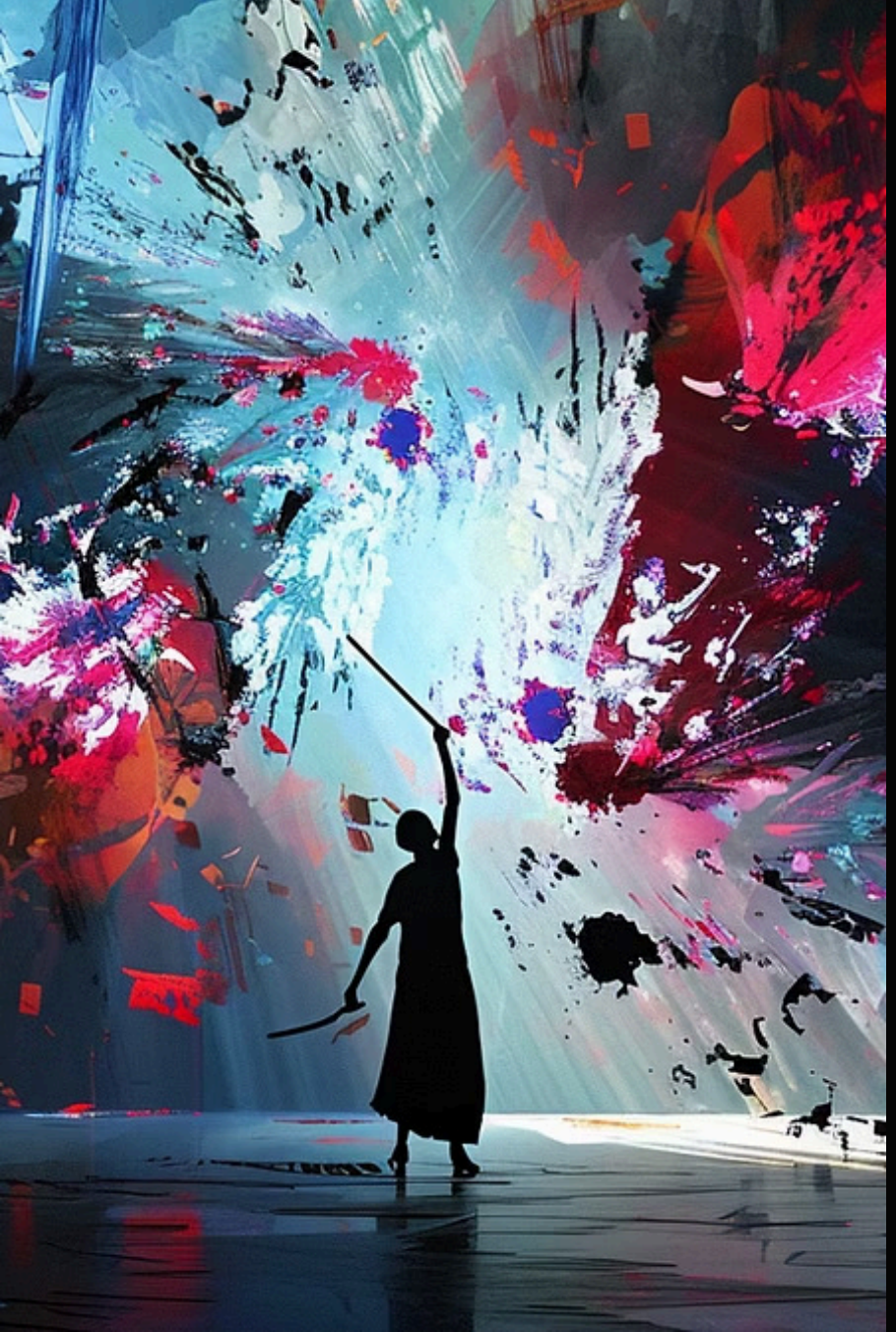


LLMOps: Taming Probabilistic Systems

Travis Frisinger

Technical Director of AI at 8th Light

8✳

About Me

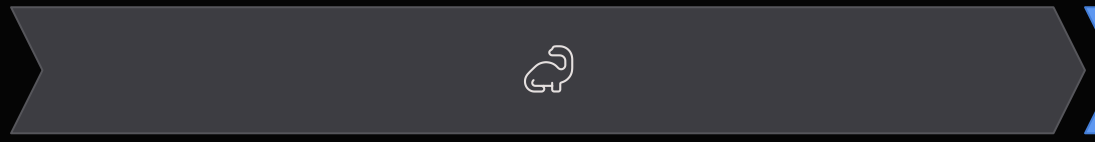
- **25 years in tech** — from Mainframe Operator to Fractional CTO
- **Leading 70 engineers** through adoption of AI-assisted development practices
- **Advising Fortune 500 executives** on bringing AI into products and rethinking how software gets built
- **Operating at the edge of practice** — architecting, coding, and mentoring alongside teams
- **Contributing to industry thought leadership** — authoring white papers, talks, and the *AiBuddy.Software* blog

How I Use LLMs

- **Partner in Thought** → Use LLMs as collaborators for strategy, ideation, and problem-framing
- **AI-Assisted Development** → Apply LLMs in the software lifecycle: coding, testing, docs, reviews
- **AI in Product** → Embed LLMs directly into customer-facing features and experiences

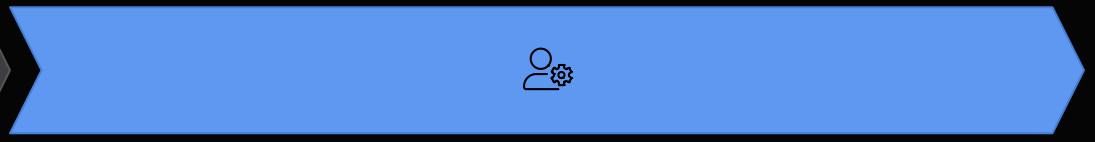
The Evolving Landscape of Operations: From Code to Cognition

The journey of building robust software has constantly evolved.



Release Engineering

The days of hand scripted deployments and long nights.

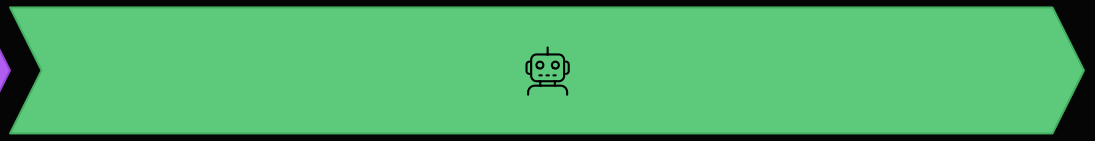


DevOps: Traditional Systems

Automating software delivery for predictable, deterministic code and infrastructure.

MLOps: Machine Learning

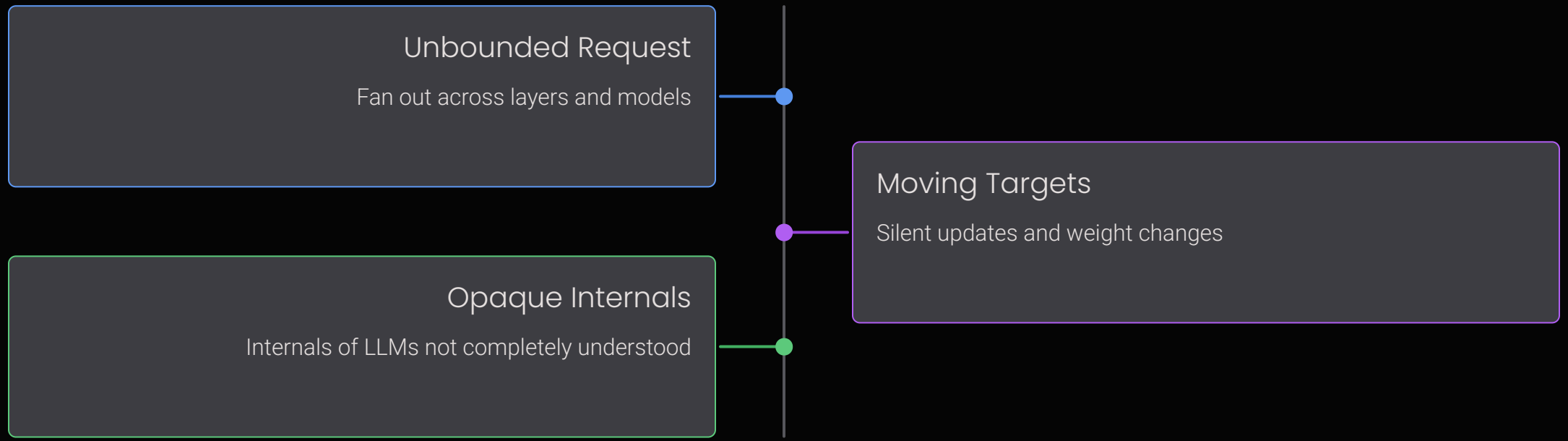
Extending practices to manage models, data pipelines, and experimental workflows at scale.



LLMOps: Large Language Models

Extending DevOps and MLOps practices to manage prompts, context, memory, and orchestration for probabilistic systems.

Why LLMs Break Classic Ops



New Ops Concerns

Model Updates

Your app changes overnight.

Hallucinations

Truth is not guaranteed. Grounding helps.

Prompt Fragility

Tiny tweaks, unstable behavior.

Compliance Surface

Prompt, inputs, context, and response all affect risk posture.

Evaluations

Testing gives way to eval pipelines.

LLMs: A New Runtime, Not Just a Tool

> **LLMs are not just tools you call.**
>
> They're dynamic, latent-space runtimes where behavior is shaped by prompt design, user context, retrieval.

From Deterministic Code to Probabilistic Cognition

How do we continue the evolution.

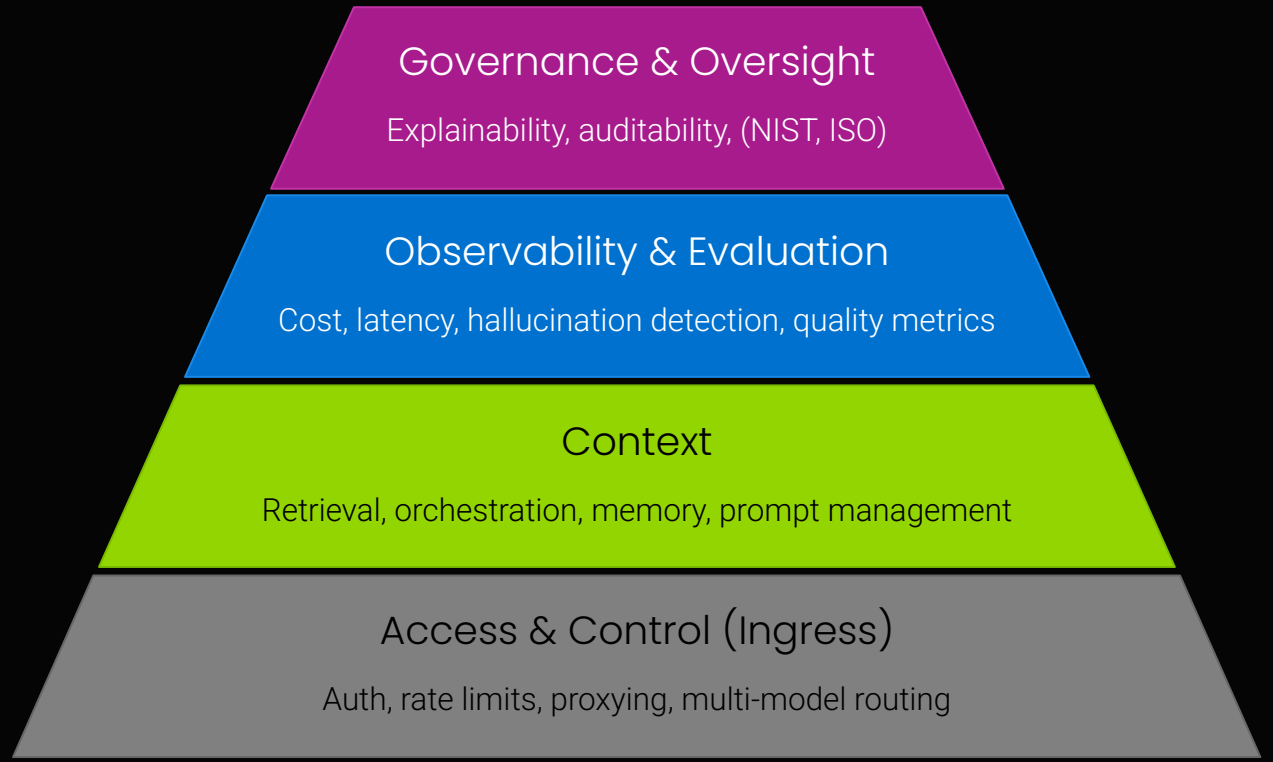
DevOps

- **Pipelines** (repeatable flows)
- **Logs & Metrics** (deterministic signals)
- **Observability** (traceable execution)
- **FinOps** (linear scaling)
- **Security & Compliance** (known boundaries)

LLMOps Adds

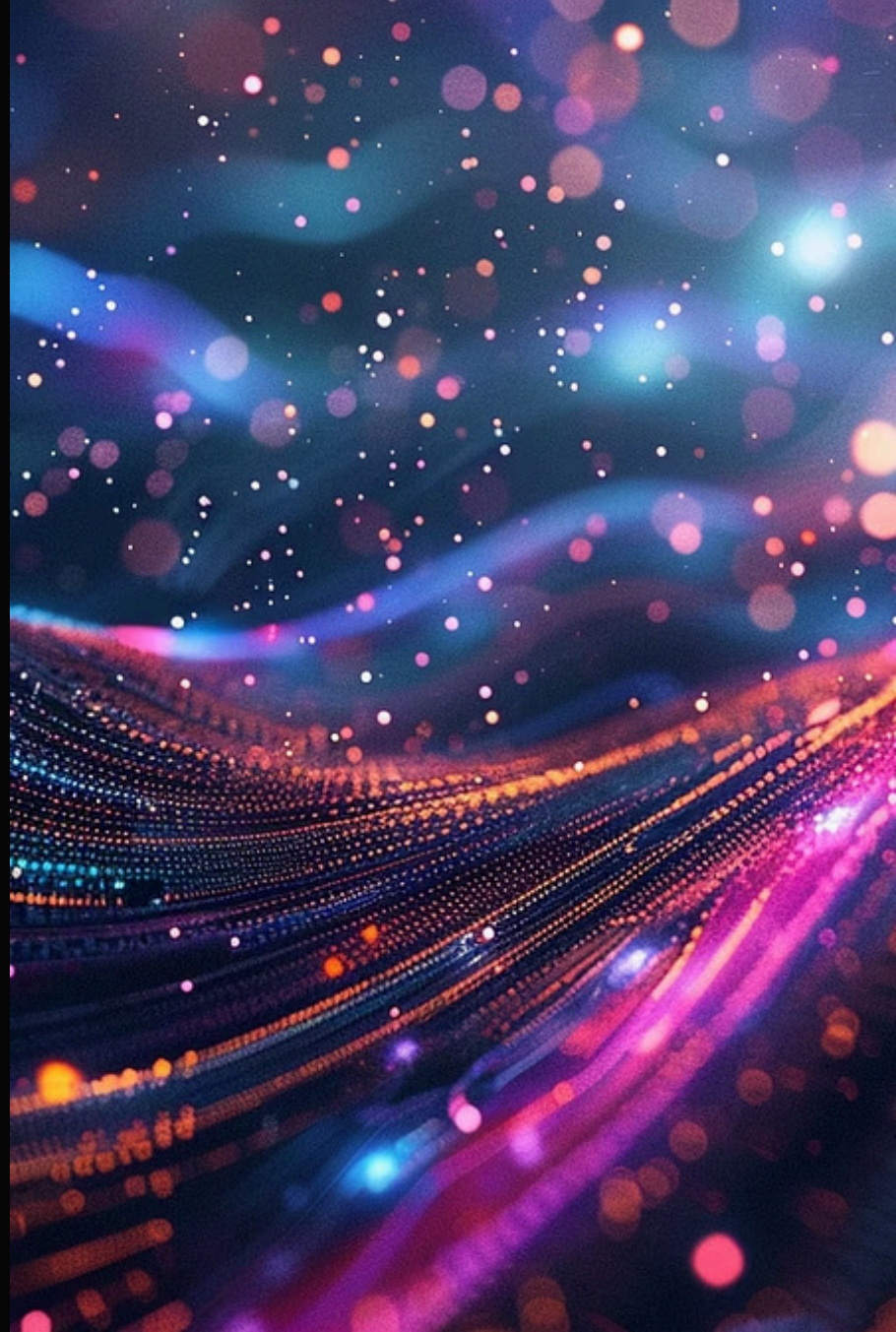
- **Prompt Management** (Language as code)
- **LLM Routing & Policy Control** (multi-model, auth, governance)
- **Hallucination Detection**
- **Latency Monitoring**
- **Evaluation Pipelines** (new CI: judges, human-in-loop, synthetic evals)
- **Context** (retrieval, memory, orchestration create heavy data dependencies)

The LLMOps Stack: From Access to Accountability



Ingress: Traffic Control at the Edge

LLM gateways provide the enforcement layer for LLM access, usage, and routing.
They're the foundation of operational control in LLM systems.
In providing services around user **authentication**, request **throttling**, and call **routing** across providers, providing **observability** hooks



LLM Gateway: Key Characteristics

Auth

- Centralized authentication and authorization for all LLM calls.
- Ensures requests are tied to identity and role (who is asking, what they can access).

Observability

- Full visibility into usage: requests, responses, latency, errors.
- Supports downstream analysis (cost tracking, model performance, data safety).

Proxying

- Acts as the traffic manager between clients and multiple LLMs.
- Enables load balancing, failover, and routing across providers or models.

Governance

- Centralized Model Authentication and Authorization
- Guardrails: filter sensitive data, redact PII, apply safety classifiers.

Portkey

Url: **https://portkey.ai/**

- ✅ **Auth & Rate Limits**: API key management, usage quotas, and policy enforcement.
- ✅ **Proxying**: Multi-provider routing, retries, fallback.
- ❌ **Context Layer**: No retrieval/orchestration built-in.
- ✅ **Observability**: Cost dashboards, latency metrics.
- ⚠️ **Governance**: Org/team-level quotas, key-level permissions.

Why:

- Strong **Ingress** (auth, rate limits, proxying) with some observability.
- Useful if your immediate need is **traffic control + cost monitoring**, not full-stack orchestration.
- Stops short at **context + governance**, so better as a building block, not a full platform.

LiteLLM

Url: **https://www.litellm.ai/**

- ✅ **Auth & Rate Limits**: API key management, usage quotas, and policy enforcement.
- ✅ **Proxying**: Multi-provider routing, retries, fallback.
- ✅ **Context Layer**: They have a request pipeline hooks to inject context.
- ⚠️ **Observability**: Logs & cost tracking, but not deep evaluation/trace.
- ✅ **Governance**: Org/team-level quotas, key-level permissions.

Why:

- Broader coverage than Portkey: strong **Ingress** plus **Governance-lite** (quotas, permissions).
- Unique **Context hooks** via request pipelines, giving a path toward orchestration.
- Still shallow on **Observability**, so not a full reliability layer yet.

TensorZero

Url: **https://www.tensorzero.com/**

- ✅ **Auth & Rate Limits**: API key management, usage quotas, and policy enforcement.
- ✅ **Proxying**: Multi-provider routing, retries, fallback.
- ❌ **Context Layer**: No retrieval/orchestration built-in.
- ✅ **Observability**: Built-in traces, latency/cost dashboards.
- ⚠️ **Governance**: Still emerging, but designed for enterprise alignment.

Why:

- Designed for **enterprise alignment**: strongest of the three on Observability and Governance trajectory.
- Provides **deep traces and cost/latency dashboards** out of the box, which makes it attractive where compliance and auditability matter.
- Aimed at **ops teams managing risk**, not just developers wiring models together.
- Best positioned if you want **production reliability and accountability hooks** rather than context experimentation.

LLM Gateway Comparison

Tool	Ingress	Context	Observability	Governance	Best Fit
Portkey	✅ Strong	❌ None	✅ Strong	✅ Strong	Dev teams needing routing + cost control
LiteLLM	✅ Strong	✅ Hooks	⚠️ Lite	✅ Strong	Teams wanting orchestration + quotas
TensorZero	✅ Strong	❌ None	✅ Strong	⚠️ Emerging	Ops teams, enterprise reliability

Prompt Playgrounds: The Forgotten Ingress

> Proxies control **who** gets in.
> Playgrounds control **what** gets in.
> They're the **missing ingress** for **prompt design.**

Prompt Playground: Key Characteristics

Playground

- Interactive space for rapid prompt iteration.
- “What you type is what you test” — immediate feedback loops.

Versioning

- **Prompts treated like code** — deployment to environments.
- Centralized prompt management

Evaluation

- Measure quality, bias, or robustness across variants.
- Turns prompt tuning from gut feel into data-driven choice.

Observability

- Strong logging and tracing — every run tied back to a version.
- Metrics on cost, latency, and output quality.

Freeplay

Url: https://freeplay.ai/

- **Playground:** Interactive prompt design and iteration.
- **Versioning:** Test packs, regression tracking, CI/CD integrations.
- **Evaluation:** Judge loops, dataset-based eval pipelines.
- **Observability:** Logging and analytics baked in.
- **Governance:** Limited, but supports feedback and QA workflows.

Why:

- **CI/CD and regression tracking** connect prompt iteration directly to engineering pipelines.
- Solid **observability baked in**, though less deep than Langfuse's trace-level detail.
- Best fit for teams who need **structured evaluation at scale** and want prompt design tied into CI/CD.

Langfuse

Url: https://langfuse.com/

- **Playground:** Live prompt testing linked to version control.
- **Versioning:** Templates, rollback, diffs.
- **Evaluation:** A/B tests, eval integrations.
- **Observability:** Strong — traces, logs, metrics tied to prompt versions.
- **Governance:** None natively, but ties into team workflows.

Why:

- Bridges **prompt design with observability**, linking prompts directly to traces, logs, and metrics.
- Strong on **versioning and rollback**, giving teams confidence to experiment without losing history.
- Best fit for teams who want **structured prompt iteration tied to metrics**, not just ad-hoc testing.

Agenta

Url: https://agenta.ai/

- **Playground:** Prompt playground with dataset-driven testing.
- **Versioning:** Prompt lifecycle mgmt, experiment tracking.
- **Evaluation:** A/B testing, multi-model comparisons.
- **Observability:** Monitoring and feedback loop support.
- **Governance:** Minimal, but designed for team use.

Why:

- Emphasizes **prompt lifecycle management**, making it broader than a one-off playground.
- Provides **monitoring and feedback loops**, but observability is lighter than Langfuse's trace depth.
- Best fit for teams seeking **end-to-end lifecycle management for prompt-driven apps**, not just ad-hoc design or isolated evals.

PromptLayer

Url: https://www.promptlayer.com/

- **Playground:** Limited — more logging-focused than interactive.
- **Versioning:** Logs every prompt + response, replay capability.
- **Evaluation:** Lightweight comparisons via SDK.
- **Observability:** Provides an audit trail of prompt usage.
- **Governance:** None — developer-focused tool.

Why:

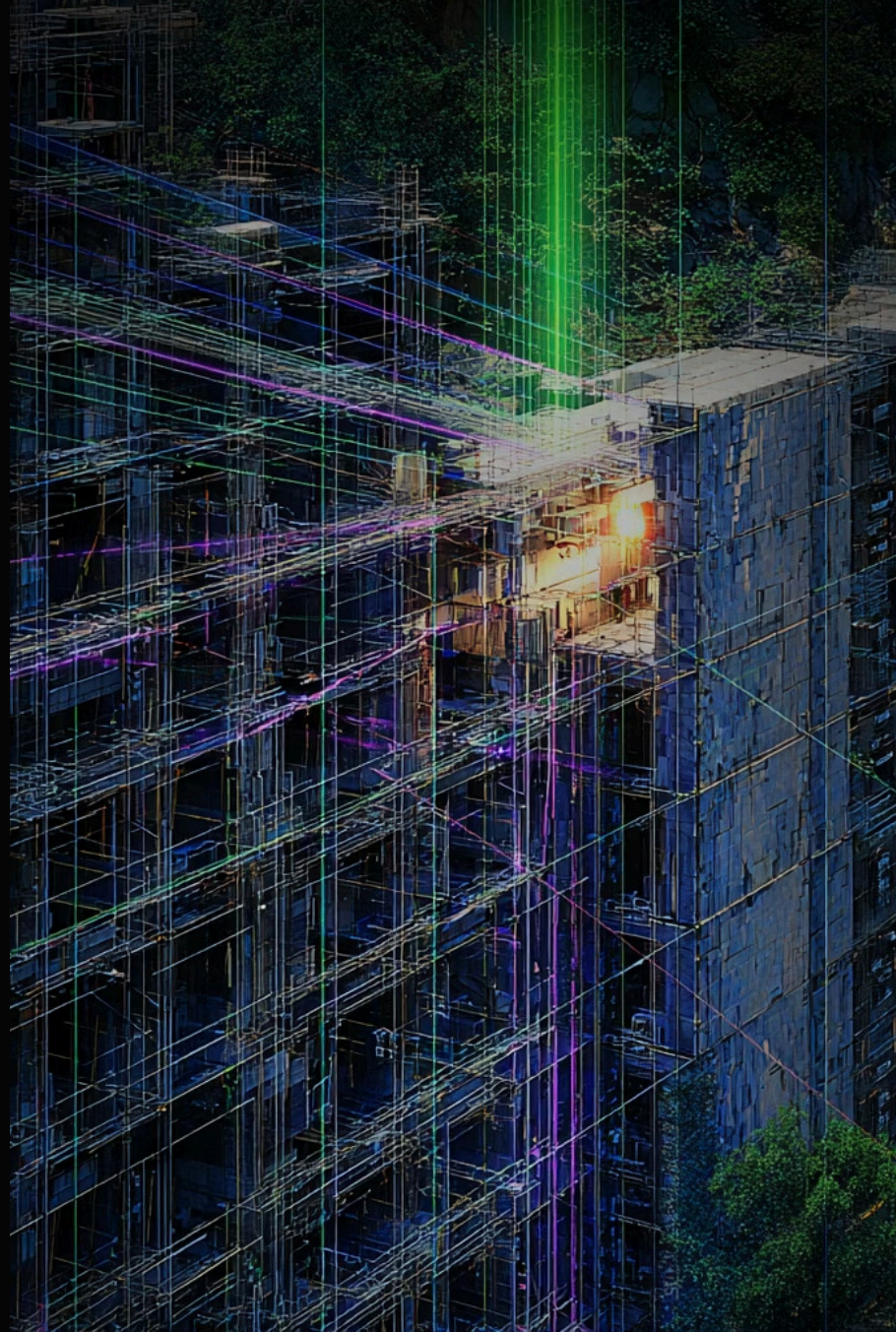
- Best suited for **lightweight audit trails**, giving teams visibility without heavy setup.
- Strong at **logging and replaying prompts/responses**, making it useful for debugging and post-mortems.
- Best fit if you need a **minimal versioning + observability layer** for prompt evolution, not a full evaluation or lifecycle platform.

Prompt Playground Comparison

Tool	Playground	Versioning	Evaluation	Observability	Governance	Best Fit
Freeplay	✅ Interactive sandbox	✅ Test packs, CI/CD	✅ Judge loops, dataset evals	⚠️ Logging + analytics baked in	⚠️ Limited (QA + feedback)	Teams prioritizing evaluation + CI/CD integration
Langfuse	✅ Live testing linked to VCS	✅ Templates, rollback, diffs	✅ A/B tests, eval integrations	✅ Strong (traces/logs tied to prompts)	❌ None natively	Teams needing observability + prompt mgmt
Agenta	✅ Playground w/ datasets	✅ Lifecycle mgmt, experiments	✅ A/B testing, multi-model	⚠️ Monitoring + feedback loops	⚠️ Minimal (team workflows)	Teams seeking lifecycle mgmt + migration path
PromptLayer	⚠️ Limited (log-focused)	✅ Logs every prompt + replay	⚠️ Lightweight SDK comps	⚠️ Audit trail only	❌ None	Devs wanting lightweight versioning + audit trail

Context: Scaffolding the LLM

> Together, the parts, **retrieval**, **orchestration** & **memory**, **scaffold** the LLM — turning stochastic outputs into usable systems.
> Without this scaffolding, *an LLM is just raw cognition* — **context** makes it **operational**.



Context: The Parts

Retrieval

Connects the model to internal and external **knowledge**

Orchestration

Structures the **workflow** and coordinates the **coherence**

Memory

Maintains **personalization** and **facts** across sessions

*These aren't libraries to be added in. They are **operational technologies** that must be managed like infra.*

Retrieval

Ops Concerns:

- **Indexing pipelines** → *Continuous ingestion & reindexing at scale*
- **Embedding lifecycle** → *Managing drift, schema evolution, and costly re-embedding cycles*
- **Scale/backups** → *Distributed infra challenges familiar from databases, now with vectors and others.*
- **Privacy & access control** → *Who can query what? Retrieval governance is as critical as model governance.*

Technologies:

- **Vector DBs** → Pinecone, Weaviate.
- **Hybrid search** → Elastic, Neo4j.
- **MCP** → *Governance-critical: auth, auditing, trusted registries required.*

👉 Everyone thinks retrieval is just plugging in a vector DB. But in reality, it's data pipelines, lifecycle management and governance. Retrieval isn't a sidecar—it's a first-class operational layer that carries all the old database headaches into the AI era.

Orchestration

Ops Concerns:

- **Workflow testing** → *CI/CD for agents and reasoning chains — keeping prompts, tools, and flows in sync.*
- **Scheduling & dependencies** → *Coordinating multi-agent workflows with temporal and data dependencies.*
- **Traceability of agent decisions** → *Auditable logs for why an agent chose a step or tool.*

Technologies:

- **Agent runtimes** (LangGraph, Autogen, CrewAI, Custom Code)

Memory

Ops Concerns:

- **TTL & pruning policies** → *Prevent runaway growth; memory isn't infinite.*
- **Snapshotting & replay** → *Enable audit trails and reproducibility for personalization.*
- **Cost/performance trade-offs** → *Balancing short-term vs long-term recall under budget constraints.*
- **Secure handling of histories** → *User-specific memory must align with privacy, compliance, and trust requirements.*

Technologies:

- **Vector DBs** reused for state (Pinecone, Weaviate, Milvus).
- **Graph DBs** (Neo4j, TigerGraph) for structured, relational memory.
- **Custom memory stores** (MemGPT, LangChain memory, Redis-style buffers).
- **Hybrid memory models** (embedding + symbolic approaches, e.g., knowledge graphs with embeddings).

👉 Memory turns ephemeral chat into persistent systems, but like any datastore, it comes with cost, governance, and lifecycle headaches.

Observability: From Black Box to Glass Box

> Observability in LLM systems means being able to **track spend by key, team, or application**, while also **tracking responsiveness across prompts and models**.
>
> It includes the capabilities to **see reasoning chains, tool use, and context applied**, and to **detect regressions, measure quality, and validate behavior** so systems remain reliable over time.

Observability : The Parts

Cost

Management by key/team

Latency

Measuring prompt responsiveness

Traceability

See the conversational chain with context

Evaluations

Detect regressions and quality failures

Cost & Latency

Ops Concerns:

- Preventing runaway spend through budget enforcement and guardrails.
- Managing latency SLAs per team, model, or endpoint.
- Attributing costs directly to prompts, teams, and workflows for accountability.

Technologies (Ingress):

- **Portkey** → Cost dashboards and latency metrics.
- **LiteLLM** → Logs with spend attribution by team/org.
- **TensorZero** → Built-in latency/cost tracing with enterprise tilt.

👉 Gateways aren't just traffic cops — they're your first observability layer.

Traceability

Ops Concerns:

- Following a request from proxy → retrieval → agent → LLM.
- Debugging: why did *this* response fail?
- Sampling vs. full logs (observability ≠ infinite logging).

Technologies (Ingress + Playgrounds):

- **Langfuse** → Traces tied to prompt versions.
- **Freeplay** → Judge loop results logged against history.
- **TensorZero** → Structured inference traces.

Evaluation

Ops Concerns:

- Regression testing on every prompt change (catch silent breakage early).
- Bias & hallucination detection feeding back into Governance pipelines.
- Automated evals in CI/CD — shifting quality checks left into delivery.

Technologies (Playgrounds + Frameworks):

- **Freeplay** → Regression packs + judge-based eval loops.
- **Langfuse** → A/B testing, eval hooks integrated into observability.
- **Agenta** → Dataset-driven prompt testing and experiment lifecycle.
- **TruLens** → Open source framework, not a platform
- **Ragas** → Another open source framework, not a platform

👉 Evaluation isn't just metrics — it's observability aimed at quality. Without it, you're shipping blind.

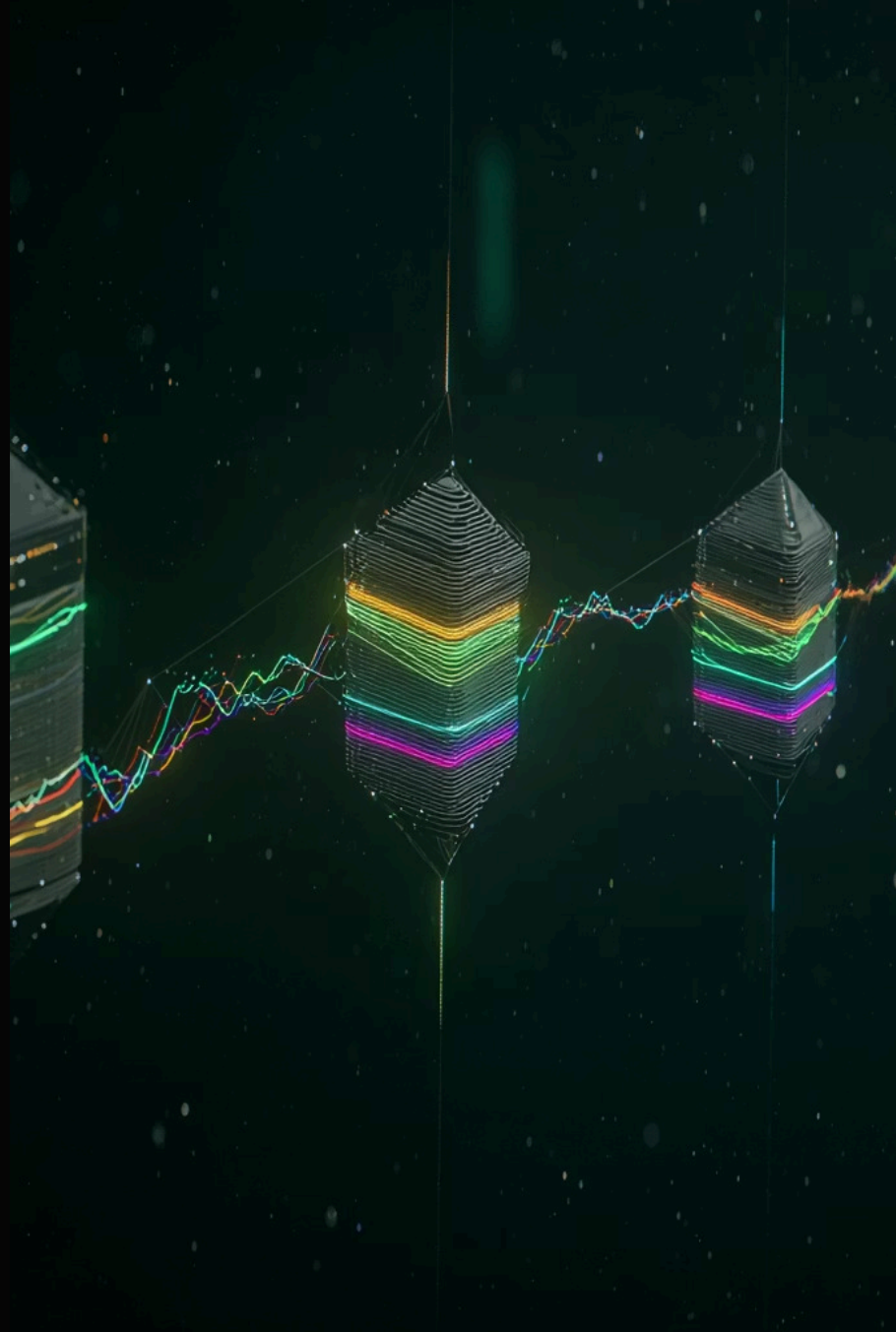
Evaluation: Measuring What Matters

Different eval types capture different failure modes — from silent breakage to bias and trust.

- **Regression** – Golden datasets, catching silent prompt breakage.
- **A/B Testing** – Comparing prompts, models, or chains head-to-head.
- **Behavioral** – Bias, hallucination, safety, jailbreak stress-tests.
- **LLM as Judge** – Scalable evals using models to grade outputs.

Governance: Oversight for LLMs

> Governance is not another dashboard — **it's the layer that decides how systems should behave,** who is allowed to use what, and whether they **align with standards of safety and fairness.**



Governance: The Parts

Permissions

Who can access which models, data, and prompts.

Risk and Bias

Detect hallucinations, toxicity, fairness issues.

Compliance

Align with NIST, ISO, internal policy.

Traceability, Explainability & Trust

Auditable chains of reasoning and decision context.

Risk and Bias: Evals, Red teaming and Guardrails

Risk and bias aren't theoretical — they surface when systems are probed, stressed, and measured.

- **Evals** – Bias, safety, robustness testing to *surface problems*.
- **Red Teaming** – Human + automated probing to *stress systems.*
  - promptfoo: **https://www.promptfoo.dev/**
- **Guardrails** – Filtering, moderation, and access limits to *contain risks.*

👉 *Evals find issues. Red teaming pushes limits. Guardrails keep systems safe in production.*

Traceability, Explainabilty and Trust

- **Traceability** – Follow the path
- **Explainability** – Make the path human.
- **Trust** – *The payoff.* When you can trace and explain, stakeholders gain confidence that the system is auditable, reliable, and aligned. Without it, AI never scales past proof-of-concept.

Key Takeaways

- **Ingress** gives you *control* (who gets in).
- **Context** makes it *usable* (retrieval, memory, orchestration).
- **Observability** makes it *measurable* (cost, latency, quality, risk).
- **Governance** makes it *trustworthy* (explainability, compliance, oversight).

👉 We've seen that LLMOps isn't about plumbing, it's about trust. From access to accountability, every layer we build — ingress, context, observability, governance — is a step toward systems people can rely on. **If DevOps was how we shipped code, LLMOps is how we'll ship cognition.**

Thank you!

Connect with me on **LinkedIn**

Visit my blog **aibuddy.software**