Agent Copilot for Model Foundry

Led cross-functional platform initiative to make model debugging faster, safer, and accessible through both UI and CLI.

PROBLEM	DELIVERABLE	IMPACT
Engineers navigated fragmented dashboards and debated ownership, metric definitions, and active versions during incidents.	A grounded observability copilot in Model Foundry UI plus a CLI for on-call and automation workflows.	Reduced time to detect and first-pass diagnosis from hours or days of manual investigation to seconds or minutes of cited evidence.

1. Leadership Context and Product Vision

As the engineering manager for Model Foundry observability, I saw a recurring operational gap: **the data existed, but engineers could not assemble it quickly enough during an incident.** Model owners had to identify the active version, move across model, feature, dataset, serving, lineage, and compliance views, then manually determine which team owned the next action.

I set the vision for an Agent Copilot that would answer operational questions in natural language while preserving platform trust: **Model Foundry metrics APIs remain the source of truth; the LLM interprets, correlates, and explains.** The same capability would be available in the web UI for interactive investigation and in a CLI for on-call engineers, scripts, and CI workflows.

2. How I Led the Program

Aligned the problem	Partnered with product engineers, model training, model serving, observability, governance, and on-call stakeholders to define the highest-value questions: unhealthy models, latency regressions, traffic drops, feature drift, stale datasets, missing lineage, and compliance gaps.
Defined trust boundaries	Required every factual claim to come from a typed tool call to an existing API. Deterministic services calculate thresholds, percent changes, freshness, and severity; the LLM clearly separates observed facts, interpretation, and recommendation.
Sequenced delivery	Started with read-only Q&A and cited evidence, then added diagnosis and next-step recommendations. Low-risk actions such as ticket creation or owner notification were approval-gated; remediation remained a later phase.
Created ownership	Established clear workstreams for UI/CLI experience, agent orchestration, metric tool contracts, evaluation, security, and operational readiness. I used shared milestones and interface reviews to prevent the agent from becoming a parallel observability stack.

3. What We Delivered

- Web copilot: contextual chat embedded in Model Foundry with entity chips, health cards, charts, tool-progress timeline, evidence links, recommendations, and approval previews.
- CLI copilot: the same typed tools and grounded answer schema exposed for on-call workflows, terminal investigation, saved queries, and automation—without requiring engineers to open multiple dashboards.
- Grounded diagnosis: model activity, latency, volume, error rate, feature availability and drift, dataset freshness, ownership, lineage, and compliance correlated into one incident-ready summary.
- Resilient execution: streaming progress, independent retries, partial answers, timeout and cancellation support, stale-data warnings, and a deterministic fallback when the LLM is unavailable.

4. Manager-Level Impact

Faster incidents	Higher leverage	More trust
Moved detection and first-pass diagnosis from fragmented, manual investigation toward seconds/minutes with a reusable evidence trail.	One shared tool and answer contract served web, CLI, on-call, product teams, and future automation instead of duplicating logic per surface.	Citations, freshness, permissions, audit logs, and explicit confidence reduced metric debate and made recommendations verifiable.

5. Technical Strategy: One Intelligence Layer, Two Surfaces

UI + CLI	Agent BFF	Orchestrator	Trusted tools	Grounded answer
Question + context	Auth + streaming	Resolve + plan	Metrics + detection	Facts + evidence

The UI and CLI both call the same conversation service and tool layer. The orchestrator resolves the model or version, fetches compact structured summaries, applies deterministic health rules, and asks the LLM to correlate the evidence. This prevents divergent answers across surfaces and lets us test tool behavior independently from model behavior.

6. Efficiency and Reliability Decisions

Concern	Decision	Why it improved the product
Latency	SSE progress + parallel independent tool calls	Users see immediate progress; model, feature, and dataset signals do not block one another unnecessarily.
Cost	Small model for intent; larger model only for complex diagnosis	Avoids paying for deep reasoning on simple lookups and keeps time to first token low.
Context	Compact summaries and evidence references	Reduces token usage and prompt risk; detailed charts are fetched only when the user drills down.
Caching	Memoize by normalized query, entity, time window, metric version, and freshness watermark	Repeated health summaries are fast, while stale cached answers are invalidated when metrics change.
Failure	Per-tool timeout, retry budget, partial-result contract, deterministic fallback	The copilot remains useful when one API or the LLM fails and never treats missing data as healthy data.
Safety	Permission-aware tools, schema validation, approval tokens, idempotency, audit trail	The model is never the authorization boundary and cannot perform an unreviewed write action.

7. Continuous Evaluation: How We Improve Answer Quality

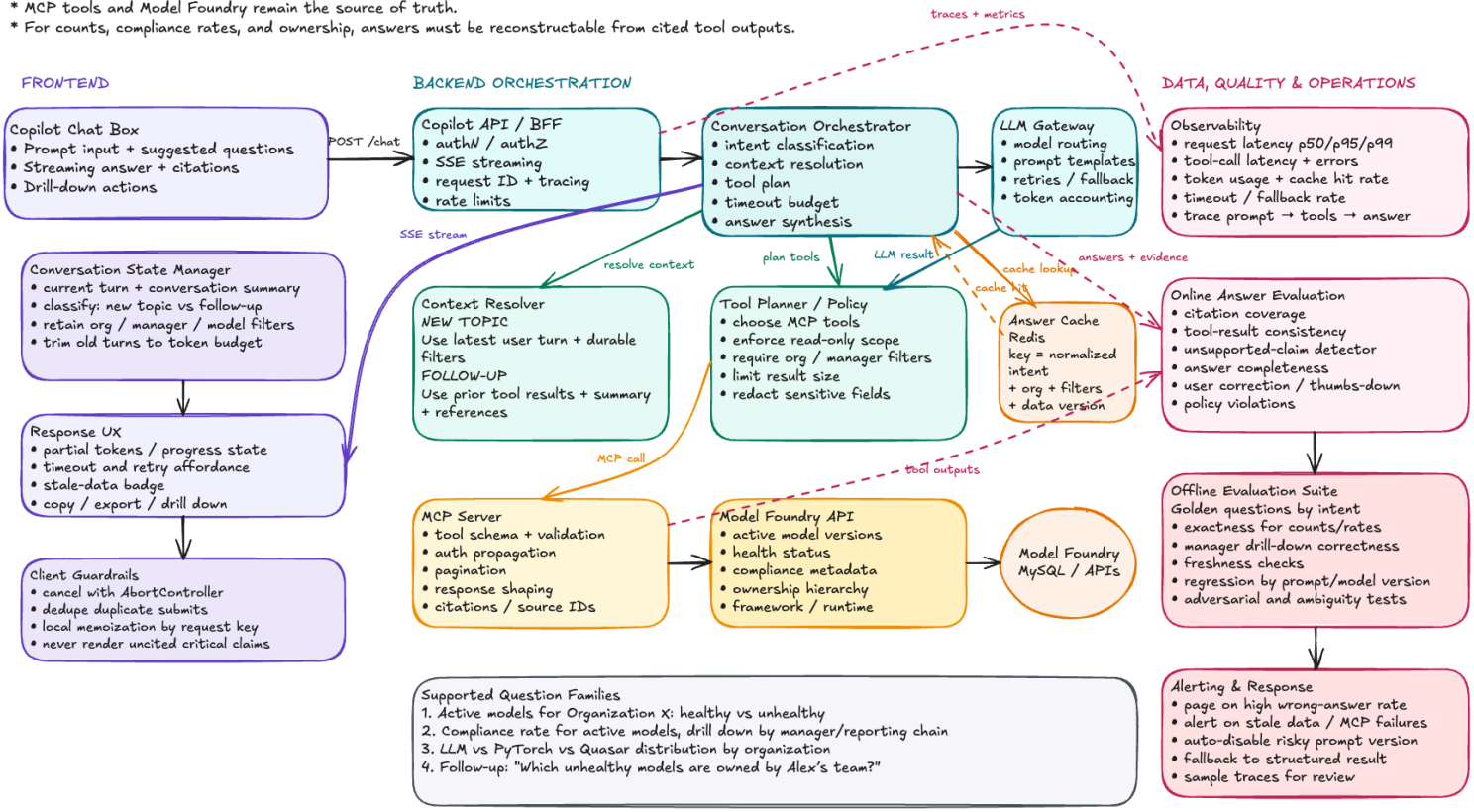
1. Observe	2. Evaluate	3. Release safely	4. Learn online
Trace every run: entity, tool calls, prompt/model version, latency, tokens, failures, evidence, and user corrections.	Replay a golden incident set and historical traces; score entity resolution, tool selection, factual consistency, citations, usefulness, and hallucination.	Use shadow mode, prompt/model canaries, regression gates, and A/B tests before broad rollout.	Measure time to diagnosis, incident resolution, recommendation acceptance, repeat usage, trust, cache hit rate, and cost per successful diagnosis.

I treated feedback as product data, not anecdote. A user correction creates a labeled evaluation case; a failed tool call becomes a contract or reliability issue; a low-value recommendation feeds the next golden-set review. We tracked quality by prompt and model version so improvements were measurable and reversible.

8. High Level Architecture Design

Model Health Copilot

* the LLM explains and synthesizes;
* MCP tools and Model Foundry remain the source of truth.
* For counts, compliance rates, and ownership, answers must be reconstructable from cited tool outputs.



Memoization strategy
- FE: dedupe identical in-flight requests
- BE: short TTL cache for stable aggregates
- Include data_version / max_updated_at in key
- Do not cache user-specific authorization decisions

Graceful Degradation and Timeout Budget
- 0–2s: show "Analyzing model health..." and stream progress events
- 2–8s: continue with partial tool results; issue per-tool deadlines
- >8s: stop expensive calls, return structured partial answer with missing sections clearly labeled
- MCP unavailable: return cached answer with freshness timestamp, or direct Model Foundry link
- LLM unavailable: return deterministic tables/charts from tool output instead of failing the request