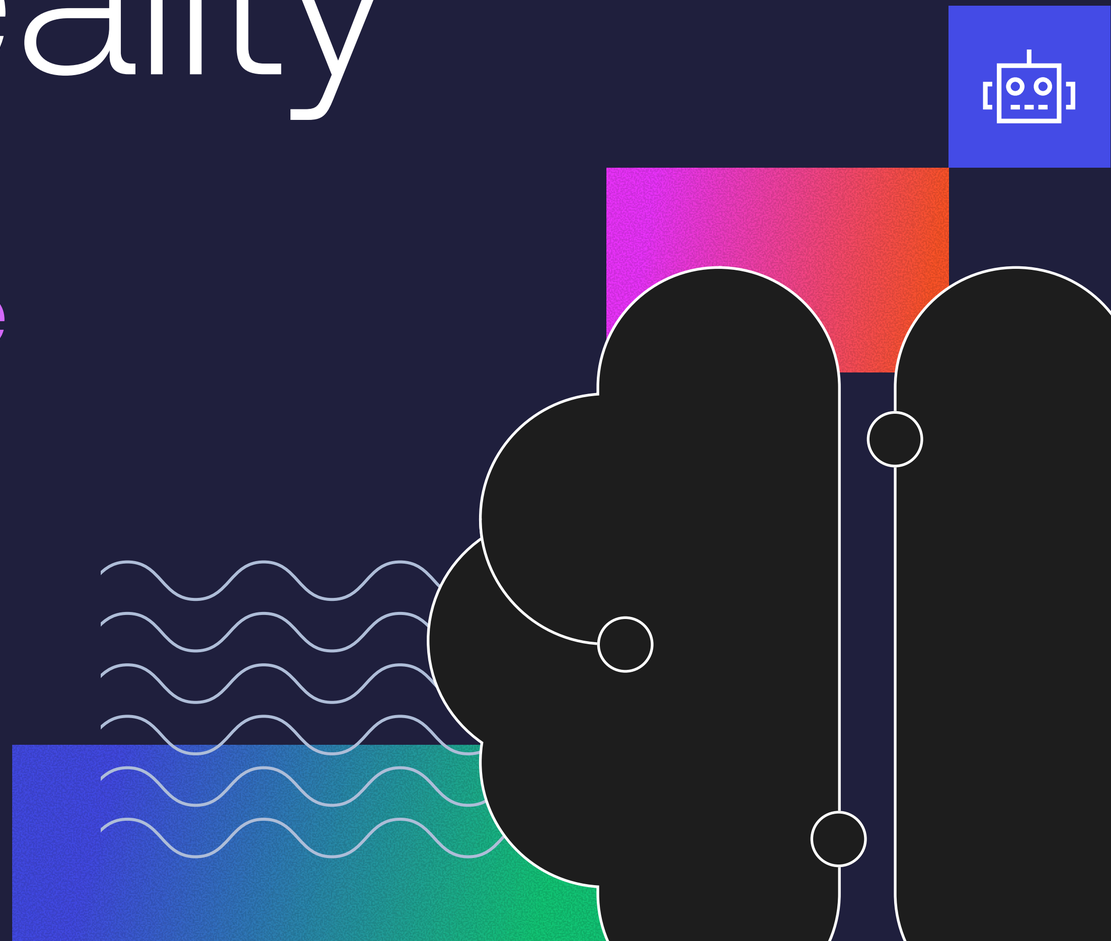
From AI Hype to Durable Reality

Why Agentic Flows Need Distributed-Systems Discipline



Temporal

I remember when I became a technophile. It was 1995, and my dad, Paul, lugged a spotted-cow Gateway 2000 box into the house. He booted Windows 95, opened Microsoft Paint, and let me explore. The moment I learned that I could clean up with a quick Ctrl+N, a world of infinite possibilities opened up in front of me, and I was hooked.

Fast-forward to a month ago. I asked my colleague Steve Androulakis how Temporal fits into the world of AI agents and Model Context Protocol (MCP). Steve sent me the “Getting Started with MCP for Claude Desktop” guide and reminded me that the best way to learn is to roll up your sleeves (Thanks Steve!).

I started with Anthropic’s National Weather Service-API example, wrapped the tools in Temporal Workflows, and turned the HTTP call into a Temporal Activity. I realized that MCP empowers LLMs to seamlessly interact with external or internal services and APIs, while Temporal fortifies these interactions with robust Durable Execution. Together, “Durable Tools” transform AI from passive responders into resilient, action-taking agents. As I watched my weather agent come to life, the same spark of excitement around infinite possibilities I felt in 1995 returned.

```
############################################################################################

# workflows.py

# Temporal's retry policies remove the need for endless try/catch blocks
retry_policy = RetryPolicy(
    maximum_attempts=0,  # Infinite retries
    initial_interval=timedelta(seconds=2),
    maximum_interval=timedelta(minutes=1),
    backoff_coefficient=2.0,
)

NWS_API_BASE = "https://api.weather.gov"

@workflow.defn # Workflow for Getting Alerts from the NWS, called from the Durable Tool above
class GetAlertsWorkflow:
    @workflow.run
    async def run(self, state: str) -> str:
        url = f"{NWS_API_BASE}/alerts/active/area/{state}"
        data = await workflow.execute_activity(
            "make_nws_request",  # Name of the registered activity, retries based on the retry_policy above,
with a 40 second timeout
            url,
            schedule_to_close_timeout=timedelta(seconds=40),
            retry_policy=retry_policy,
        )

        if not data or "features" not in data:
            return "Unable to fetch alerts or no alerts found."
        if not data["features"]:
            return "No active alerts for this state."
        alerts = [format_alert(feature) for feature in data["features"]]
        return "\n---\n".join(alerts)
```

Continues on the next page →

→ Continuing from the previous page

```
################################################################################################

# workflows.py

# Temporal's retry policies remove the need for endless try/catch blocks
retry_policy = RetryPolicy(
    maximum_attempts=0,  # Infinite retries
    initial_interval=timedelta(seconds=2),
    maximum_interval=timedelta(minutes=1),
    backoff_coefficient=2.0,
)

NWS_API_BASE = "https://api.weather.gov"

@workflow.defn # Workflow for Getting Alerts from the NWS, called from the Durable Tool above
class GetAlertsWorkflow:
    @workflow.run
    async def run(self, state: str) -> str:
        url = f"{NWS_API_BASE}/alerts/active/area/{state}"
        data = await workflow.execute_activity(
            "make_nws_request",  # Name of the registered activity, retries based on the retry_policy above,
with a 40 second timeout
            url,
            schedule_to_close_timeout=timedelta(seconds=40),
            retry_policy=retry_policy,
        )

        if not data or "features" not in data:
            return "Unable to fetch alerts or no alerts found."
        if not data["features"]:
            return "No active alerts for this state."
        alerts = [format_alert(feature) for feature in data["features"]]
        return "\n---\n".join(alerts)
```

Who I Am

I'm Kevin Paul Martin, a Senior Strategic Account Executive at Temporal who still keeps a terminal open. For the last two-and-a-half years I've guided global enterprises through their Temporal journeys, translating business goals into durable workflows.

Before stepping into the AE role, I was a Sr. Solution Architect at Redis, helping Fortune-100 teams master everything real-time. And long before that shift to "solutioning," I was a full-stack engineer at ZEFR, where I built BrandID Store — a customer-facing data-visualization web-app that armed our sales team with the insights they needed to sell targeted advertising packages.

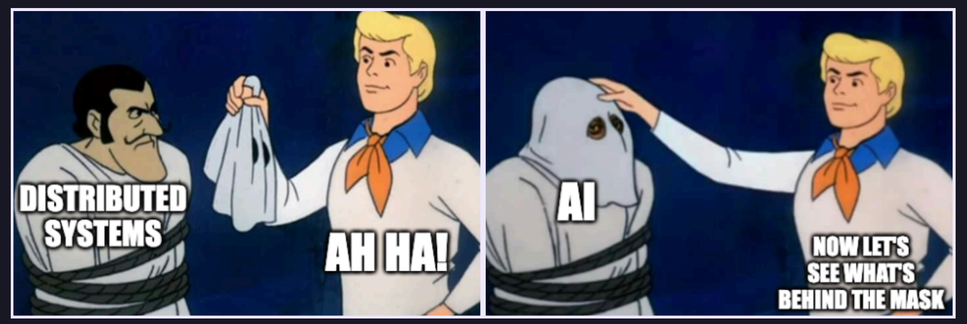
That mix of builder's curiosity and customer focus is exactly what pulled me into today's agentic-AI frontier.

Why This Post

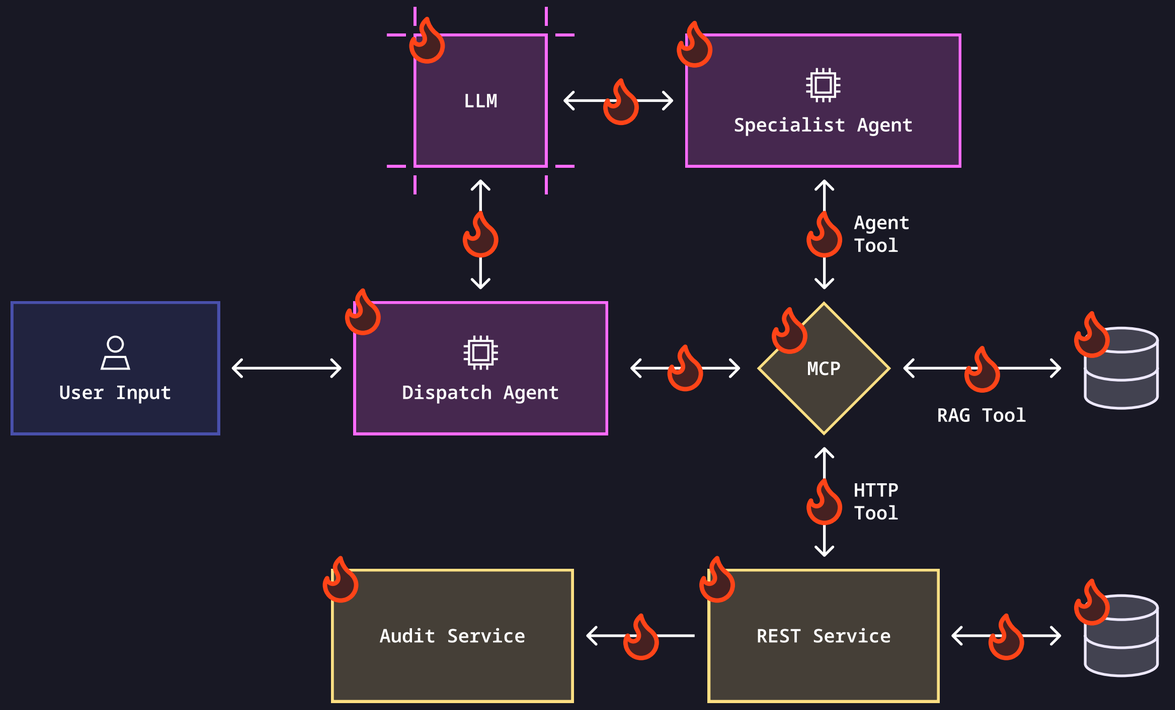
The past month of building stateful agentic flows and MCP servers taught me that the glamorous part of AI ends quickly; operationalizing for production is where projects live or die. Below are the lessons I learned while building these flows with Temporal — and why Temporal keeps showing up in the answers.

LEARNING #1

AI Systems Are Distributed Systems In Disguise

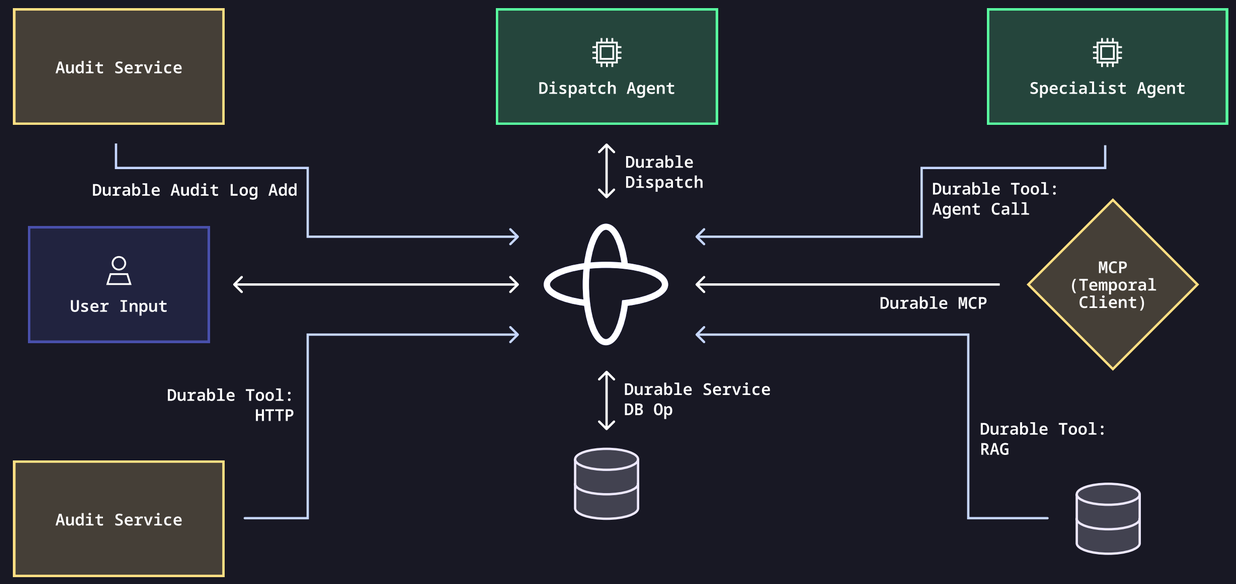


Once an AI system grows beyond a single machine, and even more importantly when you start preparing for production, its real challenges stop being “AI” and start being distributed-systems problems on steroids (see diagram to the right).



- ✓ **Scale & Parallelism —** Training or serving a large model means sharing parameters and data across GPUs, nodes, or regions — just like any other high-throughput service.
- ✓ **Data Pipelines —** ETL, feature stores, vector databases, and streaming updates must guarantee consistency, ordering, and fault tolerance.
- ✓ **Orchestration & Coordination —** Agentic, Retrieval Augmented Generation (RAG), & MCP pipelines, chain tokenizers, retrievers, LLMs, and post-processors that need idempotent retries, safe versioning, and back-pressure handling.
- ✓ **Reliability & Observability —** Health checks, retries, circuit breakers, metrics, and tracing remain mandatory; GPU kernels do not repeal the fallacies of distributed computing.

So while the logic is AI-specific, the plumbing is pure distributed systems. Temporal removes that plumbing pain (see diagram to the right): its workflow engine handles state, retries, timeouts, back-pressure, and event replay out of the box, while its built-in tracing and metrics give you instant observability — enabling teams to ship agentic flows without reinventing distributed-systems discipline.



LEARNING #2

Durable Execution Drives Resilience

Packets drop, links flap, and dependencies misbehave: the network can vanish mid-call, a downstream database might choke, a third-party SaaS endpoint may rate-limit or 500, and that “stable” vendor API you don’t control will inevitably get flaky at 2 a.m. Temporal absorbs all of it.

- ✓ Automatic retries & back-off wrap every Activity call — whether it’s an internal microservice or an external AI API—so transient failures don’t page the on-call.
- ✓ Timeouts stop hung tasks from cascading through the system.
- ✓ Event-sourced history lets a crashed worker replay state on restart: no lost progress, no double charges.
- ✓ Schedules trigger recurring or delayed actions — say, firing off a quarterly-renewal workflow; Signals let an external system inject events into that workflow — picture the CFO clicking ‘Approve’ (human-in-the-loop) in your finance portal, whose backend persists the approval to Temporal before it returns 200 OK; and Queries let downstream dashboards read the live state without mutating it. Because the approval signal is durably stored first, even if the payment-gateway API times out or the partner ERP goes offline, Temporal simply replays the workflow and keeps retrying the charge until it clears — no lost sign-offs, no duplicate invoices.

With Durable Execution, resilience isn’t an after-thought — it’s the default.

LEARNING #3

Polyglot Freedom Enables Flexibility

Each service keeps its favorite language— Python, Java, Go, TypeScript, .NET, Ruby, even PHP — while Temporal handles the cross-language coordination. A Python MCP implementation can kick off a RAG workflow, and a Java worker can perform the retrieval step; signals and queries flow seamlessly between them. This polyglot model lets AI specialists stay productive in Python while platform teams reuse existing services, making cross-functional collaboration frictionless.

LEARNING #4

Nexus Bridges AI & Business Workflows With End-to-End Observability

Nexus lets an agentic workflow invoke an existing microservice — also implemented in Temporal — through a single, durable control plane. Every hop — agent call, business logic, external API — lands in the same Workflow History and the same metrics dashboards. One system finally gives engineers, SREs, and data scientists the unified view they've wanted for years, eliminating the need for separate schedulers, sidecars, or observability silos. AI and non-AI paths share durability, tracing, and alerting in one place — and platform complexity shrinks accordingly.

LEARNING #5

Crash-Proof Conversations: Signals, Queries & MCP In Action

Steve's demo is my favorite proof-point: using Signals and Queries, a chat workflow stays fully stateful even if the user slams the laptop lid. When they reopen it, Temporal replays the entire event history and the conversation picks up exactly where it left off — no "Sorry, I forgot what we were talking about."

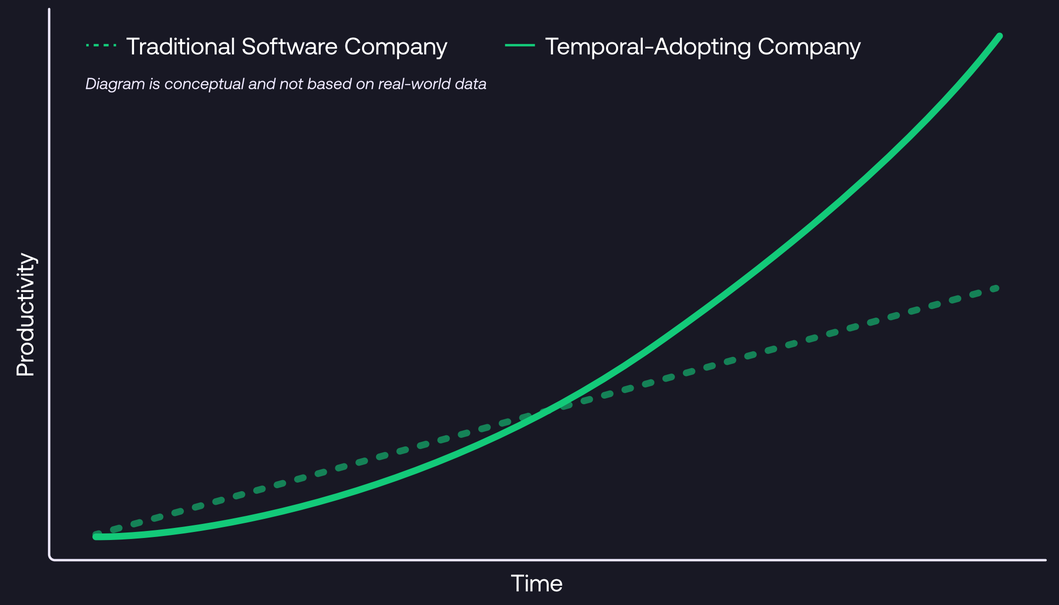
I saw the same durability in my own MCP demo. The MCP server acts as a Temporal client, and every tool invocation is a StartWorkflow call. If the process hosting the agent crashes mid-dialog, the Workflow History simply rehydrates on the next Worker, the Signal channel reconnects, and the agent finishes the request without losing context or duplicating work.

Bottom line: whether it's Steve's multi-turn chat or my MCP toolchain, Temporal turns fragile conversational loops into crash-proof, replayable workflows.

LEARNING #6

Developer Velocity Through Code Simplicity

Durable Execution, implicit retries, and built-in state management slash boilerplate. Teams often report double-digit reductions in lines of code and a dramatic drop in custom glue logic. Fewer moving parts mean faster prototypes, quicker reviews, and easier audits.



> "Without Temporal, we would be lagging behind significantly from where we are. Because of the productivity it provides, we are able to move much, much faster than where we were previously. I don't think that the product I work on would be successful at all if Temporal were not part of the solution."
>
> — Rob Zienert, Sr. Software Engineer, Infra Management, Netflix

LEARNING #7

"Durable Tools" Means Free Scalability

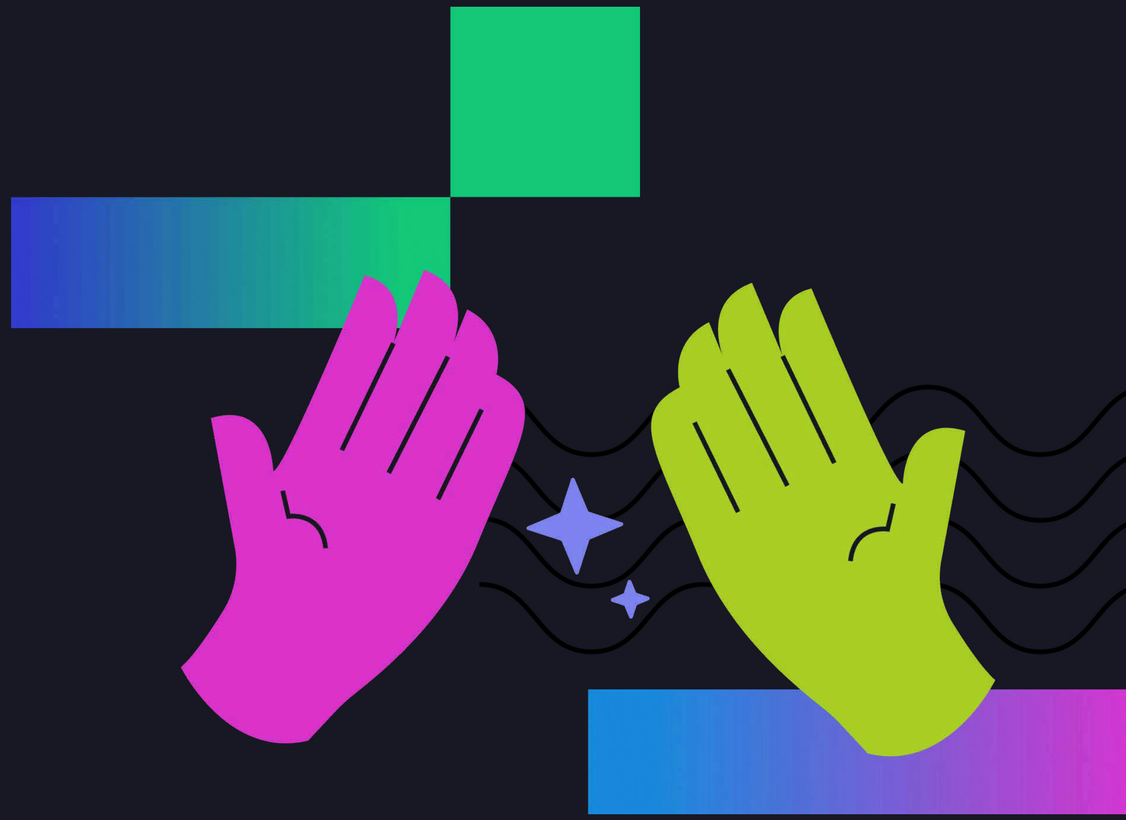
When each MCP tool is implemented as a Temporal Workflow, instead of running inside the MCP server process, every invocation is picked up by the global worker fleet — the same pool that already scales your non-AI workloads. That architecture delivers horizontal elasticity without extra infrastructure:

- ✓ A burst of agent requests simply fans out across additional workers (or k8s autoscaled pods) with no changes to the MCP server code.
- ✓ Hot spots are isolated: slow or compute-heavy tools don't block lightweight steps, because workers pull tasks independently.
- ✓ Back-pressure is automatic: the Temporal Server queues pending tasks, so the MCP server never collapses under peak load. In practice, this "Durable Tools" pattern lets a single MCP server stay thin and stateless while Temporal handles the heavy lifting of concurrency, throughput, and resource isolation — scaling right alongside the rest of your platform.

Social proof

> “Temporal is used for asynchronous workflows and operations inside OpenAI. Temporal is a neat workflow solution that makes multi-step workflows reliable even when individual steps crash, without much effort by developers. It’s particularly useful for longer-running workflows like image generation at scale.”
>
> **— Gergely Orosz**

The quote above comes from OpenAI’s post on The Pragmatic Engineer blog, which states that Temporal was employed to manage asynchronous operations and ensure reliability during the launch of ChatGPT's image generation feature. Temporal's workflow orchestration capabilities were particularly beneficial in handling the high demand and complexity associated with large-scale image generation tasks. TPE is one of my favorite industry blogs, and I highly recommend subscribing if you haven’t. Additionally Temporal powers OpenAI’s Codex product, a powerful agentic tool for coding quickly.



Takeaways & How to Turn Hype into Durable Reality

Agentic AI is exhilarating, but durability, scalability, observability, and developer velocity decide who reaches production. Temporal lets you focus on prompts and models while it handles the distributed-systems discipline underneath.

If you're experimenting with MCP reliability or agent orchestration, let's connect.

- Join the Temporal Community Slack (channel #topic-ai).
- Find me on LinkedIn & GitHub.
- Sample Code: temporal-invoice-mcp, temporal-durable-mcp-weather-sample.
- Or email sales@temporal.io and mention "AI blog" in the subject line.

We can help you turn the AI hype into durable reality — by giving every agentic flow the distributed-systems discipline it deserves. Try your hand at improving your agentic AI production with a free trial of Temporal Cloud with $1,000 in credits.