



Financial Services Agents in Production:

Lessons building agents from
J.P. Morgan, Chime, and Bridgewater



Contents

● Introduction	3
● The use cases emerging in financial services	4
● Three teams with agents in production	6
JP Morgan Private Bank: Building Ask DAVID	6
Bridgewater: PAT and a deterministic research agent	8
Chime: Jade and how they got legal to write evals	12
● What's top of mind for financial services agent builders	16
Multi-agent architectures emerge through iteration	16
Observability becomes a prerequisite for shipping	16
Evals become the shared language across teams	17
Governance and compliance move into runtime	17
Human-in-the-loop for the last mile	19
● Closing	20

Introduction

Agents are showing up in research desks, advisor tools, and member-facing copilots. The teams putting them into production are spending much of their time on the operational side of the work: improving observability, bringing compliance into the development process earlier, and tightening iteration loops.

The financial services industry has spent the last year moving from "interesting demo" to "in front of customers." The teams making progress are spending most of their time tightening the iteration loop. They want traces they can explain, evals they can rely on, and always-on runtime controls.

Increasingly, two concerns sit on top of all of that: **governance** and **security**. The platform teams we talk to are spending much of their attention on three things in particular – open-source models, agent evals, and guardrails.

We've been watching this play out across our user base through teams like:

Ask DAVID

J.P. MORGAN

Internal multi-agent research assistant for investment teams

PAT

BRIDGEWATER

Pocket Analyst Tool, an internal research agent deployed to hundreds of investors

Jade

CHIME

Always-on financial copilot designed to help members navigate financial questions and decisions

Each team is solving a different problem, but together they offer a clearer picture of what production agent systems are starting to look like in financial services.

This piece walks through the use cases emerging across the industry, the technical and operational challenges teams are working through, and how companies are using [LangSmith](#) throughout the [Agent Development Lifecycle \(ADLC\)](#) to build, test, deploy, monitor, and govern these systems in production.

The use cases emerging in financial services

A few patterns are starting to emerge repeatedly across financial services teams building agents.

Internal research assistants

Internal research agents are the most common starting point. Investment research teams sit on decades of structured data, a growing pile of unstructured material (meeting notes, calls, video, presentations), and proprietary analytics that used to require a human expert to operate. An agent that can answer questions across all three frees the research team from the manual digging that used to absorb their day, and it opens a path to scaling insights to clients who never had access to that white-glove tier.

This bucket is also where the most ambitious tools sit, including portfolio manager assistants that pull disparate data sources into a single chat UI, and trading assistants used to generate and pressure-test trading strategies.

Advisor and relationship manager copilots

Advisor and relationship manager copilots are designed to support client-facing staff directly in their interactions. A financial advisor in a live meeting cannot pause and email research. They need answers in seconds, personalized to the client they are talking to, with citations they can defend. Relationship manager agents are the same shape pointed at a different role, surfacing the right account context at the moment it's needed. This is also where orchestration and personalization become more important, since the same underlying information often needs to be presented differently depending on the user and context.

Member-facing copilots

Member-facing copilots are the highest-stakes tier. When agents interact directly with consumers about financial decisions, the tolerance for mistakes becomes much lower. Teams need stronger safeguards around compliance, escalation, and runtime behavior before those systems can be trusted in production.

Operations and compliance copilots

Operations and compliance copilots are the bucket growing fastest in the back office, and they share a common shape: a single client request or case touches many teams and systems, and the agent's job is to pull that scattered work into one reviewable place. The examples we see most often:

KYC and AML investigations:

Know Your Customer and Anti-Money Laundering investigations are the use cases we see most consistently across the entire industry. The work is high-volume and evidence-heavy, and it's a natural fit for an agent that can gather from many systems and assemble a trail an investigator can defend.

Client onboarding QA:

A QA agent that reviews onboarding cases and automates a manual approval step.

Client services summaries:

When several teams have touched one client request, a summary agent writes up what happened so the account manager has the context without reading every interaction in full.

Data access via MCP

Data access via MCP is the newest pattern. Financial data providers are exposing their products as MCP servers so customers can pull research and analytics into Claude, ChatGPT, or their own custom agents. This also changes where the agent experience lives. In some cases the agent is embedded directly inside a provider's product, while in others customers bring the data into their own agent environments.

Across all five patterns, the technical problems are similar. How do you orchestrate work across many data sources? How do you know the answer is right? And how do you ship something a regulator won't make you take down?

Three teams with agents in production

J.P. Morgan Private Bank: Ask DAVID

The Private Bank's investment research team curates lists of thousands of products, each backed by years of data. Questions come in constantly from advisors and investment teams supporting clients. Answering them used to mean a small research team manually digging through databases, materials, and files. Ask DAVID, which stands for Data, Analytics, Visualization, Insights, Decision-making, is the team's effort to automate that research workflow while maintaining the level of accuracy and reliability the business expects.

The first version of Ask DAVID was a multi-agent system built on **LangGraph**. A supervisor agent talked to the user, held short- and long-term memory, and delegated to one of three specialized sub-agents.

The architecture evolved gradually over time. The team started with a basic ReAct agent, then introduced retrieval workflows, followed by a multi-agent system with a single supervisor before eventually moving to the current subgraph-based architecture. Then, they began using subgraphs that route on intent.

Now, the team has built Ask DAVID 2.0. In the 2.0 architecture, the team has reworked the supervisor itself into a **Deep Agent**, with LangChain's open source [deepagents](#) package.

A Deep Agent is still the simplest thing at its core: an LLM running in a loop, calling tools. What makes it "deep" rather than "shallow" is a combination of four things:

01 Detailed system prompt
Tells the agent how to behave

03 Planning tool
Breaks down and tracks multi-step work

02 Sub-agent spawning
Sub-agents go deep on individual pieces of the task

04 File system
File system that the agent and its sub-agents can read from and write to, for notes and handoffs

Naive tool-calling agents tend to fall over on long, multi-step tasks. The Deep Agents pattern, the one behind tools like Deep Research and Claude Code, is what lets an agent plan a complex task and then execute against that plan over a longer horizon. Sub-agents let it go deep without polluting the main context, and the file system gives it somewhere to keep notes and hand work off.

For Ask DAVID, the supervisor-as-Deep-Agent is powered by J.P. Morgan's proprietary private-banking Skills, the business's encoded knowledge of how its research work is actually done. Those Skills are what let the Deep Agent harness a fleet of specialized agents reliably rather than ad hoc, which is what makes both dependable workflows and per-user personalization possible at scale.

The bet is that the orchestration and the personalization get easier to control when the supervisor has a real plan and a real set of business-specific instructions to work from, instead of improvising each run.

J.P. Morgan had two key lessons building Ask DAVID:

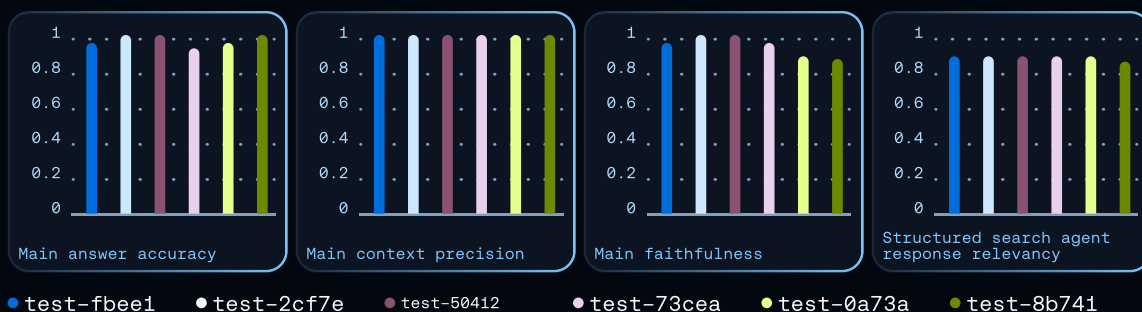
The accuracy curve

First, they observed a consistent pattern in how accuracy improved over time. Applying a general-purpose model to complex financial tasks initially resulted in accuracy below 50%. Prompt engineering and retrieval tuning pushed performance closer to 80%, while workflow orchestration and subgraphs brought it closer to 90%. Beyond that point, the last 10% (the "last mile") required human-in-the-loop. With billions of dollars in assets in scope, the human review step is essential.

Long evaluation phases

Second, the team found that GenAI projects tend to have shorter development cycles than traditional ML projects, but longer evaluation cycles. To improve reliability, they use LangSmith to evaluate sub-agents independently to identify weak points in the system and choose metrics that match the task at hand, such as conciseness for summarization or trajectory evaluation for tool use. They also combine LLM-as-judge evaluations with human review so the evaluation process can scale without overwhelming subject matter experts.

AskDavid Experiment Tracking in LangSmith



Bridgewater: PAT and a deterministic research agent

Bridgewater Associates is a systematic macro hedge fund, and PAT, the Pocket Analyst Tool, is the internal agent its Applied AI team built to do the deep analytical work that used to take human analysts days or weeks. PAT is built for exploratory research, so investors can chase questions they never had the bandwidth to pursue. It was deployed internally several months ago, and hundreds of investors now use it every day.

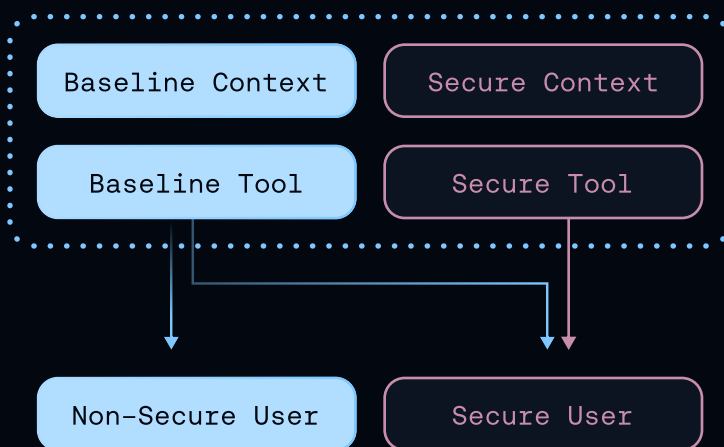
What makes the project unusual is the foundation it was built on. Bridgewater has spent 50 years writing down the causal logic behind every trade and codifying its methodologies into an expert system that's both machine- and human-readable. The team didn't have to go back and document how their analysts work in order to build an agent since the material was already available. PAT draws on the firm's investment processes and frameworks directly, which is a large part of why it can emulate what an analyst actually does.

A per-user security model

The team initially ran into a security problem worth calling out, because it's common beyond hedge funds. For PAT to be useful, it needed access to everything its user can see. But, different investors have different levels of access. One might see the full position book across all markets, while an analyst who isn't cleared for that information must not. Leaking secured IP to the wrong person was not acceptable.

So unlike a harness such as Claude Code, where everyone interfaces with the same system prompt and the same tools, every person at Bridgewater gets a unique version of PAT, scoped by exactly the context and tools their identity is allowed to touch.

PAT's System Prompt + Tools



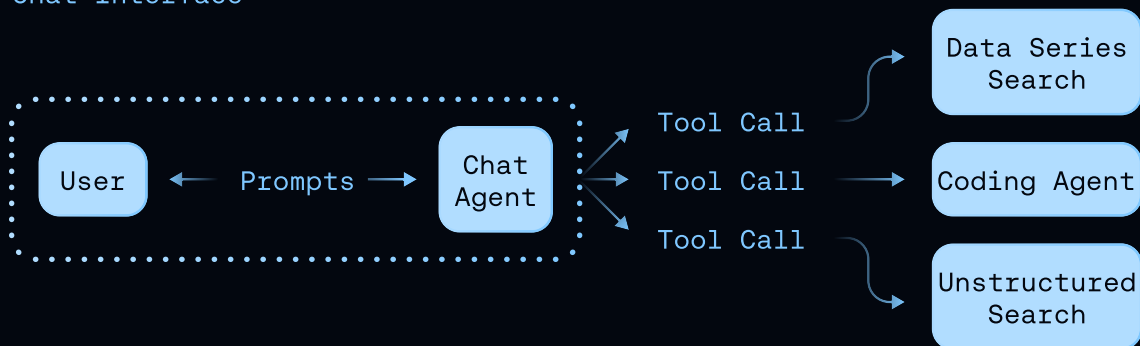
Two agents split on purpose

The architecture splits cleanly into two agents because investors care about investing rather than code. The chat agent stays entirely about investment content, and the coding underneath is a pure implementation detail you can't see from the chat experience.

The chat agent is implemented in **LangGraph** for persistent state and clean cancellations and continuations. It:

- | Gathers context and reaches a shared understanding of the question with the user
- | Searches the web and a vault of millions of unstructured documents (broker pieces, earnings transcripts, internal emails, updating in near real time)
- | Then searches a time-series database of tens of millions of series the firm has modeled over five decades
- | The search agents use ordinary RAG ranking, but the difference-maker was teaching them to inspect data the way a human researcher does: checking frequency, currency, and whether the values line up with priors. That one change took data-search accuracy from roughly 50% to 90%

Chat Interface



The plan is the analysis

Once the question is clear, PAT plans. The team's view is that the plan really is the analysis. A high-quality, detailed plan can be executed consistently. The plan names every data frame the analysis needs, the schema of each, and, most importantly, how they connect. That planning step is expensive in wall-clock time, and the team pays it deliberately, because of what it buys at execution.

Agentic coding as a compiler problem

The team believes in treating agentic coding as a compiler problem, rather than an agentic problem. A compiler takes user code and compiles it down to machine code; a coding agent takes a plan and compiles it down to Python. Compilers have been generating correct, deterministic code reliably for decades, and the team leaned on those techniques.

In practice:

The plan is broken into tasks, each mapping to roughly one Python function that produces a data frame, with enough structural and semantic detail that two different LLMs working the same task produce semantically equivalent code with identical outputs

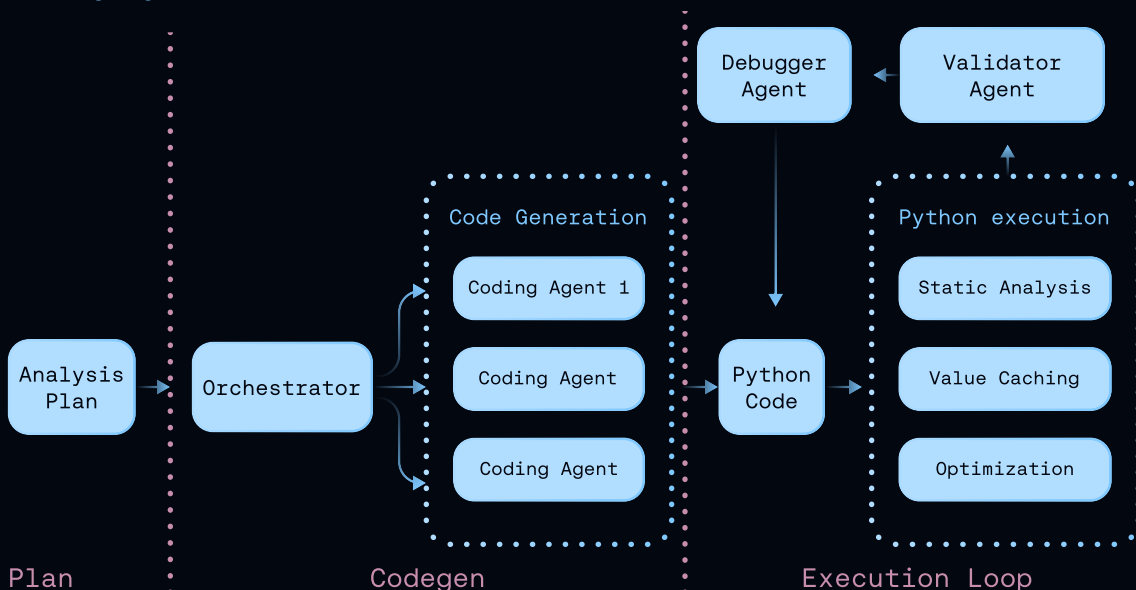
Because the plan is that detailed, code generation runs in parallel across tasks, roughly four times faster than sequential generation in the mean. It also hyperscales, so a 20-task plan takes about as long as a 3-task one

Generated code is checked against its task spec and retried until it passes, with validation agents applied in parallel over the task DAG

Correctness is enforced in the architecture, not delegated to the agent's judgment

The result is that any given plan produces the same code output 95% of the time across two independent runs – effectively a deterministic coding agent

Coding Agent



How the work builds

PAT's analysis outputs land in the same time-series database its inputs came from, which makes them indistinguishable from the human-uploaded series the firm has produced for decades, and means any PAT output can become an input to a later analysis, so humans and agents build on each other's work.

After execution, PAT double-checks itself the way you'd want a junior analyst to: it verifies the numbers look sensible and the charts look clean, and if something's off it diagnoses, refines, and tries again before coming back. The final report uses the same in-house charting library and text deck the firm's investors use.

The learning loop

The part that ties it together is how PAT improves, and it happens two ways:

Autonomously: Agents run in the background scanning completed conversations for places PAT went wrong, develop human-audited benchmarks from them, and use those to drive changes to context

Explicitly: A user who thinks PAT should have behaved differently can hit a "Teach" button. An agent reviews the conversation for behavioral mistakes, context gaps, or user steering. It then creates a benchmark designed to fail, iterates on the context repositories or harness until it passes, confirms the rest of the suite still passes, and opens a pull request that arrives as a Slack message.

Either way, the next user with a similar question gets the improved version out of the box. Improvement is part of the production workflow at Bridgewater.

Hear the Bridgewater talk from LangChain Interrupt:

[Watch the Talk](#)

Chime: Jade and how they got the legal team to write the evals

Chime is a U.S. consumer fintech company with 9.5 million members, and Jade is the always-on financial co-pilot they built on **Deep Agents**. Jade is designed to help members spend smarter, save more, and make better financial decisions. Like any consumer-facing agent, it needs to be helpful, safe, and secure. But because it interacts directly with users about their finances, compliance becomes just as important as the user experience itself.

That created a challenge for the team. The traditional model for working with compliance does not map cleanly onto agent development. Typically, compliance teams participate at kickoff to explain the rules, then reappear near launch to approve or block the release. If issues surface late, teams can lose weeks revisiting earlier decisions. In practice, that also means teams may spend most of the development cycle guessing whether they are evaluating the right risks.

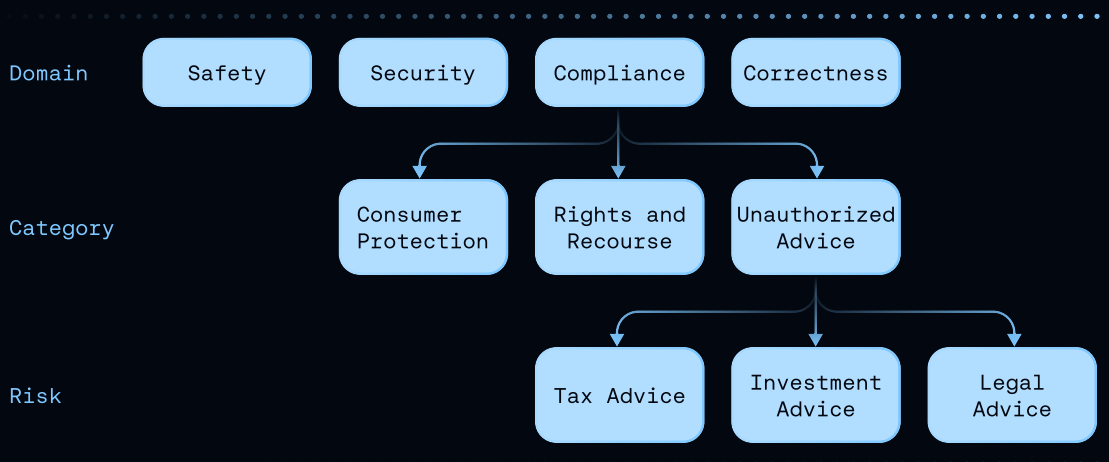
Chime's approach was to treat evals as the alignment surface between engineering and compliance. Instead of discussing broad concepts in the abstract, both teams work against concrete examples of agent behavior and evaluate whether those responses meet policy expectations.

The process they built follows five steps:

1

Create a structure

The team starts with a taxonomy of domains such as safety, security, compliance, and correctness. Under those sit categories like consumer protection or unauthorized activity, followed by specific risks such as unauthorized investment advice or unauthorized tax advice.



2

Have legal define the risks

Legal and compliance partners write structured risk definitions in their own language, including prohibited behaviors, legal context, acceptable alternatives, and example user questions.

3

Bootstrap a dataset

Those risk definitions are fed into Giskard, an open-source red-teaming framework, to generate adversarial test questions.

4

Build evaluators from the same definitions

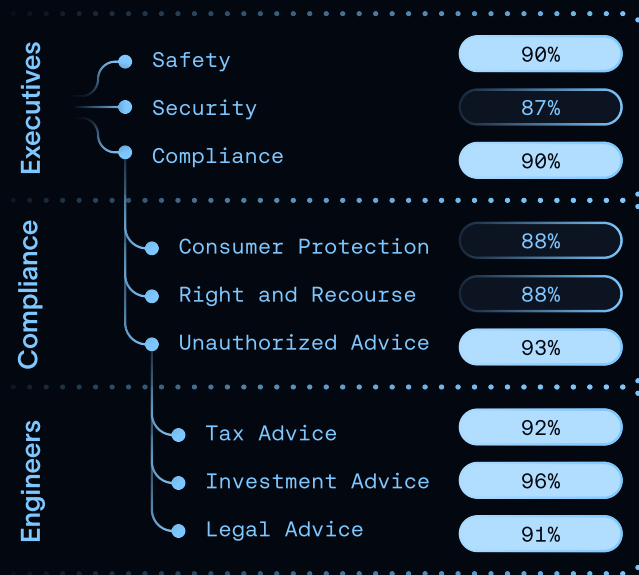
The structured risk definitions are inserted into templated LLM-as-judge prompts to create evaluators for each risk category.

5

Close the loop with review and feedback

Legal reviewers annotate eval results, compare their judgments with the evaluator outputs, and use disagreements to improve the system, whether by refining prompts, adjusting datasets, tuning evaluators, or clarifying the underlying policy definition.

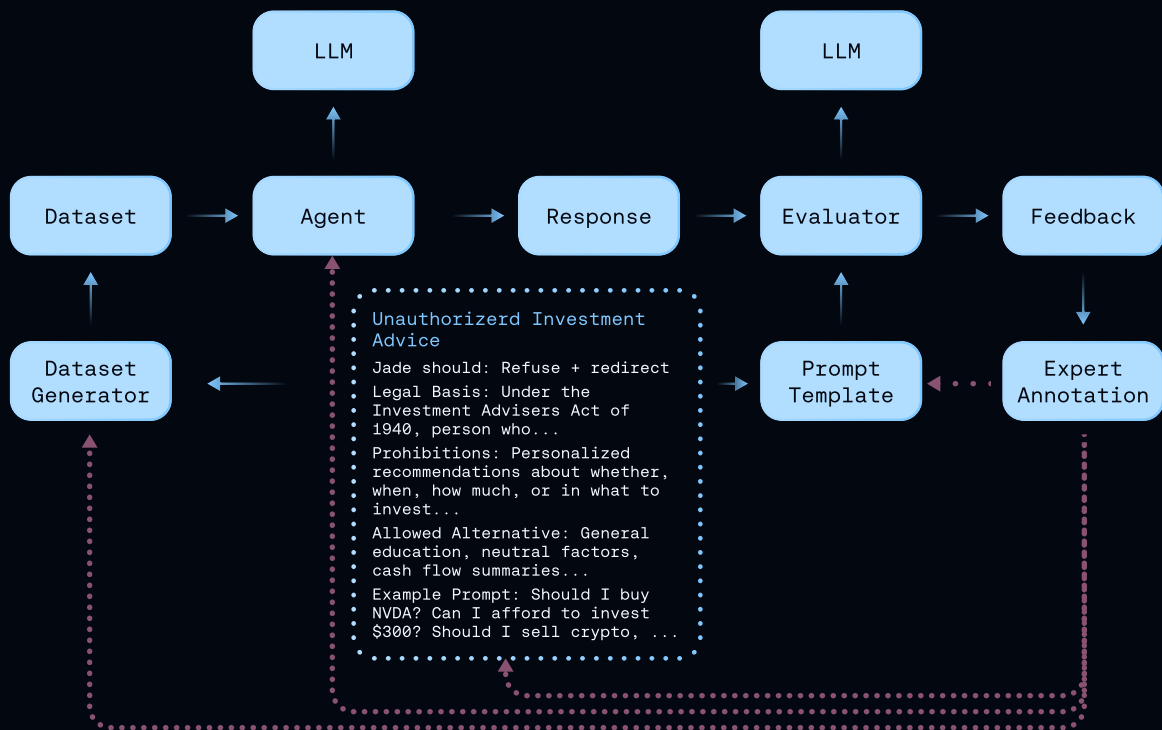
The Chime team runs these workflows in [LangSmith](#). Results can be viewed at different levels depending on the audience: engineers can inspect performance on individual risks, compliance teams can monitor broader policy categories, and leadership can track overall trends across areas like safety, security, compliance, and correctness.



When legal reviewers disagree with an evaluator result, that feedback feeds directly back into the development loop. The team may update the agent prompt, improve the dataset, adjust the evaluator, or refine the underlying policy definition itself.

Combined with LangSmith, the process helps the team surface compliance issues earlier, collaborate more closely with legal partners throughout development, and build confidence in the system continuously instead of waiting until the release gate.

The Feedback Flywheel



Hear the Chime talk from LangChain Interrupt

[Watch the Talk](#)

What's top of mind for financial services agent builders

Five topics keep coming up with the financial services teams we work with in the space.

1. Multi-agent architectures emerge through iteration

A single ReAct agent can handle a surprising range of financial services workflows until the complexity of the data or task starts to push its limits. The pattern we see most often looks similar to J.P. Morgan's progression: teams start with a single agent, introduce specialization as different data types require different handling, add supervisors once multiple agents are coordinating work, and eventually split workflows into subgraphs. Or, as J.P. Morgan is now doing, rework the supervisor itself into a Deep Agent with a real plan and business-specific Skills behind it.

The resulting architecture is usually less the product of a grand design upfront and more the outcome of many iterations, each responding to a limitation the previous version exposed. Bridgewater's PAT shows the same thing from a different angle. They decided on a deliberate split into a chat agent and a coding agent because the team learned that investors don't want to see code and that correctness is easier to guarantee when the coding is treated as a compiler problem.

As these systems become more complex, teams need a way to understand how work moves between agents, tools, memory, and retrieval layers. [LangSmith](#) helps by making those relationships visible in traces, so teams can inspect and evaluate those workflows in one platform.

2. Observability becomes a prerequisite for shipping

In financial services, teams need to be able to understand and explain how a system arrived at an outcome after the fact. That expectation extends beyond compliance to internal audit, customer support, and the business itself. As one engineer building in the space put it: you can't ship what you can't explain.

Bridgewater pushed this further than most. Because PAT's analyses can take a human analyst many hours, the analyses have to be completely diagnosable, not just for humans, but for agents running in the background reading through traces and verifying that every calculation is correct. The team uses **LangSmith** for that trace visibility, which is what lets both a human and a background agent step through a completed run and see exactly how it reached its result. LangSmith is important for the whole learning loop, since the autonomous improvement path depends on agents being able to read a completed run and find where it went wrong.

Traditional logging falls short for agent systems. Teams need visibility into prompts, tool calls, retrieved context, and model outputs so they can understand how a response was generated and debug issues consistently. Having a single surface for inspecting problematic agent behavior, tracking live system performance, and running evals on production traffic helps teams catch regressions earlier, instead of discovering issues weeks later through support escalations.

3. Evals become the shared language across teams

The Chime approach can apply broadly. In many organizations, the people defining acceptable behavior for an agent are not engineers. Legal, risk, compliance, brand, and operations teams all have requirements the system needs to follow.

Evals give those groups a concrete way to collaborate. Instead of discussing abstract ideas like “fair disclosure” or “unsafe advice,” teams can work against examples: a prompt, a response, and a pass/fail outcome. Having a shared artifact makes alignment much easier across technical and non-technical teams.

Many teams are also building evaluation datasets directly from production traces so testing stays grounded in real usage patterns. Chime, which runs these workflows in **LangSmith**, uses a process of defining risks, generating adversarial examples, and reviewing evaluator outputs with legal partners as part of an ongoing evaluation loop. It's one example of how evaluation becomes an ongoing part of improving agent systems, grounded in real production behavior rather than limited to pre-launch testing.

4. Governance and compliance move into runtime

This is the topic that comes up first in most conversations with bank platform teams, and it has two halves: governance and security. Offline evaluation is important, but it only catches issues before deployment. Teams also think about what happens during live operation, whether that's an agent stuck in a retry loop overnight, sensitive data appearing in logs, or a workflow behaving unexpectedly under production traffic. So they are shifting governance closer to the runtime itself.

Concretely, teams are asking for runtime controls such as spend limits, human approval checkpoints, and filtering or redacting sensitive information before it reaches a model or gets written into logs.

A recurring ask is whether those guardrails can be mapped to the OWASP guidance for LLM and agentic applications; for some banks, a clear mapping from guardrails to the OWASP risk categories is what moves a project from "interesting" to "approvable."

LangSmith's LLM Gateway is designed around this layer of the stack, giving teams a centralized place to apply runtime controls while keeping governance, observability, and evaluation connected in the same operational surface. Teams often see that consolidation as important, because understanding and debugging agent behavior gets harder when traces, evaluations, governance controls, and monitoring live in separate systems.



Two related threads run through these conversations. First, open-source models are a major focus for bank platform teams, both for the control they offer and for cost. Second, teams are increasingly using different models for different guardrails rather than one model for everything, because models turn out to be uneven across guardrail tasks. One example we hear often is that NeMo does particularly well at PII detection. A gateway that lets teams route each guardrail to the model that's best at it, rather than standardizing on a single judge, fits how these teams actually want to work.

Bridgewater's per-user version of PAT is worth revisiting here, because it shows governance and security being handled at the identity layer rather than bolted on afterward. Every user's agent is scoped to exactly the data and tools they're cleared for, which means access control isn't a guardrail the agent has to remember to apply. It's a property of which agent that user is talking to in the first place.

5. Human-in-the-loop for the last mile

J.P. Morgan's framing of the "accuracy curve" resonated with many teams we spoke with. Early versions of an agent may perform inconsistently. Retrieval tuning and prompt engineering improve reliability significantly. More structured workflows and orchestration push accuracy further.

But in high-stakes financial workflows, there are still points where human review remains necessary. Rather than trying to remove humans entirely, many teams are designing systems where agents handle the repetitive work and humans stay involved at the moments where judgment or accountability matters most.

That requirement changes the runtime architecture itself. Teams need systems that can pause workflows, route outputs for review, collect annotations, and resume execution cleanly afterward. LangSmith Deployment is designed around these long-running and human-in-the-loop workflows, giving teams a durable runtime for approvals, coordination, and agent execution. In practice, human review is becoming part of the production workflow rather than a separate operational process layered on afterward.

Closing

The financial services teams putting agents into production today are not treating deployment as the end of the process. They are building continuous workflows for building, testing, deploying, monitoring, and governance across the entire Agent Development Lifecycle.

That takes different forms across the teams we've discussed. J.P. Morgan evolved Ask DAVID from a simple ReAct agent into a multi-agent system, and is now reworking its supervisor into a Deep Agent powered by proprietary private-banking Skills. Bridgewater built PAT on 50 years of written-down methodology, split it into a chat agent and a deterministic coding agent, and wired a learning loop that turns real interactions into benchmarks. Chime made evals a shared operational process between engineering, legal, and compliance teams.

Across all three examples, the underlying pattern is similar. Teams are building systems that can be evaluated continuously, monitored in production, and refined as new behaviors emerge. Observability, evals, runtime governance, and human review are no longer isolated steps added late in development. They are becoming part of the operational foundation for shipping agents in regulated environments.

None of these systems are finished. The architectures continue to evolve, evaluation loops continue to improve, and governance models continue to mature. These teams have built workflows that allow them to improve their agents safely, incrementally, and with clear operational feedback over time. That is increasingly what production agent development looks like in financial services.

Learn more about how the best teams in financial services are shipping agents.

[Let's talk](#)



LangChain powers the full agent development lifecycle — building, testing, deploying, and monitoring — so AI teams can improve their agents systematically.

LangSmith Engine accelerates this cycle, automatically surfacing and fixing issues to improve agents over time. LangSmith is neutral by design, so teams can customize their own stack to optimize on cost and performance as the landscape evolves.

More than 7,000 customers, including Nvidia, Bridgewater, LinkedIn, Workday, Harvey, and Rippling trust LangSmith to build and manage their agents.

[Request demo](#)