Agents

Part II: How to Design an Agent

Architectures, Protocols, and Technical Evaluation

Michael J Bommarito II · Daniel Martin Katz · Jillian Bommarito

December 20, 2025

Working Draft Chapter

Version 1.01

This chapter is Part II of a three-part series from the textbook *Artificial Intelligence for Law and Finance*. Part I (What is an Agent?) provides definitions and foundations. Part III (Chapter 08 — Agents Part III: How to Govern an Agent) addresses regulation, risk, and deployment.

The most current copy of the project is available at:

`https://github.com/mjbommar/ai-law-finance-book/`

Contents

1 Introduction

In *What is an Agent?*, the first part of this series (Bommarito et al. 2025), we introduced the GPA+IAT framework—a way to recognize agentic systems through Goals, Perception, Action, Iteration, Adaptation, and Termination. Now, we turn to the practical: *How do you design one?*

The answer starts with a simple observation: agentic systems are meant to do real work—augmenting human professionals, automating routine workflows, or even replacing entire organizational functions. Whatever the ambition, the system must handle the same work those humans and organizations currently perform. A contract negotiation agent must do what a transactional lawyer does; a portfolio monitoring agent must "act like" an analyst. This means an agentic system requires the same structural capabilities as a professional team—from receiving and understanding work to coordinating action and operating under governance controls.

The Core Insight

Agents are not magic; they are architecture.
Behind each architectural question is a design decision with real tradeoffs. Every capability that makes an agent useful corresponds to a concrete choice about how the system should work, and those choices shape what the system can do, how reliably it performs, how it fails, and what controls you have over it.

Notably, agents are also more than any single agentic framework, programming language, or foundation model. While the implementation of an agentic architecture does require choosing between languages like Python or C# and models like GPT or Claude, the design decisions transcend these details.

While we encourage you to learn by doing, we also encourage you to begin by focusing on the core concepts and design decisions in this chapter first. Many commercial software providers and even open source libraries obscure or confuse the key design decisions that underlie agents, and you will likely learn best by building your own foundation before learning the language of any particular vendor or framework.

This chapter organizes those architectural decisions into ten questions—the kind you should be asking whether you're building, evaluating, deploying, or governing an agentic system.

1.1 The Ten Questions

Designing an agent means answering these ten questions. The capabilities an agent needs are not determined by the technology; they are determined by the work. And work has structure that any

system, human or artificial, must accommodate.

Consider how a law firm operates. Work arrives through defined channels: a client calls, court filings appear on the docket, or an originating attorney refers a matter to a specialist. That work typically arrives as instructions that need clarification, and associates quickly learn to read between the lines, developing heuristics for what partners actually want rather than what they literally said. Research demands access to the right databases and thoughtful search strategies. Actions like filing motions or sending client letters have real consequences and require appropriate authorization. Institutional knowledge accumulates in case files and precedent databases. Complex matters break down into workstreams with dependencies, and work products have clear completion criteria. Associates know when to escalate to partners, teams coordinate across practice groups, and compliance controls keep the whole operation within ethical and regulatory bounds.

A discretionary portfolio management team follows the same pattern. Market data and research flow through defined feeds, and analysts must interpret investment committee mandates that leave room for judgment. Research requires access to financial databases, company filings, and market intelligence. Trades have real-world consequences that demand compliance checks before execution. Position history and investment theses persist across quarters, informing future decisions. Portfolio construction breaks down into sector allocation, security selection, and risk management, each with its own completion criteria. Analysts escalate to portfolio managers when positions approach limits, teams coordinate across asset classes, and regulatory controls ensure fiduciary compliance throughout.

These structural parallels are not coincidental. Law firms and investment teams are both *cognitive work systems*: organizations that process information, make decisions, and take consequential actions under uncertainty (Hollnagel and Woods 2005; Hollan et al. 2000). Agentic systems are cognitive work systems too (Wang et al. 2024a; Rao and Georgeff 1995), which means they face the same architectural challenges and require the same structural capabilities.

This mapping has practical implications. When you evaluate an agentic system, you can ask the same questions you would ask about a professional team. When you design governance for an agentic system, you can draw on the same frameworks that govern professional organizations. When you communicate with technical teams, you can use organizational language they will understand.

Table 1 lists these ten questions in the order an agent encounters them during execution.

Table 1: Architectural questions for agentic systems

Section	Architectural Question
Triggers	How does the agent know when it has work to do?
Intent	How does the agent understand what is being asked?
Perception	How does the agent find things out?
Action	How does the agent make things happen?
Memory	How does the agent remember things?
Planning	How does the agent break a big job into steps?
Termination	How does the agent know when it is done?
Escalation	How does the agent know when to ask for help?
Delegation	How does the agent work with other agents?
Governance	How do we design systems that can be governed?

Each section addresses one question through organizational analogies, architectural concepts, domain-specific considerations for law and finance, and governance implications. You can read sequentially for cumulative understanding, jump directly to whichever question matters most, or skip to the end for synthesis.

Lastly, remember that, like other human processes or software systems, agents require governance. Our next chapter, *Governing Agents*, addresses compliance frameworks and controls in detail.

2 How Does an Agent Know When It Has Work to Do?

Consider again a day in the life of an average attorney. How does work reach their desk? Typically, a client calls with a question, a court docket updates with a new filing, a calendar reminds them of an upcoming motion, or a junior associate knocks on their door with a question that exceeds their expertise. Agentic systems operate in the same way. No matter how sophisticated, a system with tools, memory, and planning capabilities remains idle until work "arrives"; the architectural question is how these triggers and channels are defined and designed.

Triggers

Triggers are the events that start agent execution. In practice, a trigger might be a docket alert, a price crossing a threshold, a calendar deadline coming due, or an internal "I can't proceed safely" signal from the agent itself. Without a trigger, even a highly capable system sits idle.

The distinction between triggers and channels matters for system design. A trigger is the *what*: the event that demands attention. A channel is the *how*: the pathway through which that event reaches the agent. The same trigger, such as an approaching deadline, might arrive through different channels depending on context. It could come from a calendar system, an email reminder, or a human prompt asking "what's due this week?" Understanding this separation helps you design systems that can receive work from multiple sources while maintaining consistent processing logic.

Triggers also create the audit trail that governance depends on. Every action an agent takes traces back to the trigger that initiated it, and when a regulator asks why the system flagged a transaction, or when a court demands production of the agent's reasoning, you need to show what event started the chain of analysis. Systems that cannot trace actions back to triggers cannot be meaningfully audited, and in highly regulated fields like law and finance, what cannot be audited likely should not be deployed. We formalize this requirement as part of the logging architecture in Section 11.1.

Channels

Channels are how triggers reach the agent. In professional practice, four channels cover almost all work intake:

External feeds: The world pushes work to you (court filings, market data, regulatory updates).

Human prompts: People request work directly (chat, email, collaboration platforms).

Scheduled jobs: Time itself triggers execution (deadlines, periodic checks, end-of-day).

Escalation events: Internal signals that ask for human help (budget exhaustion, low confidence).

Each channel type serves a different operational need. External feeds enable reactive monitoring, allowing the system to respond to events as they occur in the world. Human prompts enable interactive collaboration, letting professionals direct the system's attention as needs arise. Scheduled jobs enable proactive workflows, ensuring that routine tasks happen reliably without requiring someone to remember to initiate them. And escalation events close the loop on human oversight by ensuring the system asks for help when it reaches its limits, a topic we explore in depth in Section 9.

The underlying point is simple but easy to overlook: before an agentic system can reason or act, it must first notice that work exists. Channels are the sensory apparatus of the system, the means by which it becomes aware of its environment and the tasks waiting to be accomplished. A lawyer cannot respond to a redlined contract they never received, and an agent cannot act on a trigger it never observed.

Triggers and channels are architectural concerns—questions about how work reaches the system, how events are routed, and how actions are logged for audit. But architecture alone does not determine whether an agentic system succeeds in practice. Professionals must actually use the system, and

that raises a different set of questions: through what interaction modality do users engage? Is the experience synchronous or asynchronous? Does the system push information or wait to be asked? These are user experience questions, and they center on the concept of *surfaces*.

Consider the difference between a litigator who explores docket alerts through an interactive research session, refining queries as understanding develops, versus one who receives a daily summary email, versus one who gets a polished memo only when something requires attention. The underlying trigger is identical. The channel is the same. What differs is the surface—and surface design is the province of UX professionals, not system architects.

We address surfaces in this section because they complete the picture of the human-agent interface. Triggers, channels, and surfaces together determine whether an agentic system fits naturally into professional workflows or feels like an awkward intrusion. System architects must understand surfaces to design appropriate triggers and channels; UX designers must understand triggers and channels to design effective surfaces. The disciplines are distinct but interdependent. After examining how work arrives through each channel type, we turn to how users experience the system through different surfaces (Section 2.5).

2.1 External Feeds: The World Pushes Work to You

External feeds deliver events from systems outside the agentic system's direct control. The external system pushes notifications when events occur, much like receiving service of process rather than checking the courthouse daily to see if you have been sued.

Legal and Regulatory Feeds: Court docket systems and state e-filing platforms can send notifications whenever documents are filed in cases you are monitoring. When an alert arrives, an agentic system can retrieve the filed document, analyze its contents, and trigger the appropriate response, whether that means flagging a motion for attorney review, updating a case timeline, or drafting a preliminary response memo. SEC filing systems offer similar capabilities for corporate disclosures, enabling agentic systems to monitor competitors' filings and flag material changes. Citator services can notify the system whenever cases you care about are cited or overruled, keeping your legal research current without manual checking.

Financial Market Feeds: Financial institutions receive real-time market data through providers like Bloomberg and Reuters. A portfolio management agent can subscribe to price alerts and receive notifications when thresholds are crossed, then evaluate whether rebalancing is warranted and either execute trades within pre-approved limits or escalate to a portfolio manager for approval. News feeds add another layer, delivering headlines, earnings announcements, and analyst ratings that the agent can assess for materiality and surface to managers when developments warrant attention.

Figure 1: Four channel types through which work reaches agentic systems. External feeds push events from outside systems; human prompts arrive through interactive interfaces; scheduled jobs trigger on time-based conditions; and escalation events signal internal limits requiring human intervention. All channels converge on the agentic system's event router.

Speed vs. Reasoning: A Critical Distinction

Market data arrives at millisecond granularity. LLM-based reasoning operates at second-to-minute timescales. This mismatch means agentic systems are unsuited for high-frequency trading or latency-sensitive execution, but well suited for strategic decisions, research synthesis, and compliance monitoring. The architecture pattern: fast deterministic systems handle real-time capture and threshold detection; the agentic system processes only after an alert is raised.

Reliability Matters: How external feeds connect to agentic systems has real consequences for governance. Some integration approaches prioritize speed but may occasionally miss events; others guarantee delivery and maintain complete logs but add latency. For regulated applications, you generally want the latter: systems that ensure every event is recorded, processed in order, and available for later audit. When evaluating vendor systems, ask whether the integration approach

guarantees delivery and maintains audit trails, or whether events can be lost without detection.

2.2 Human Prompts as Events

Human prompts feel different from external feeds because they are interactive. You type something and expect a response. But at the architectural level, a human prompt is just another event type: the user generates an event, the system receives it through a channel, processes it, and responds. Treating prompts this way simplifies design, because all events flow through the same routing and prioritization logic rather than requiring separate handling for interactive versus background work.

Chat interfaces offer the most direct channel for human interaction, and they are where most professionals first encounter agentic systems. A litigation associate researching a motion to dismiss might type "Find Ninth Circuit cases on personal jurisdiction over foreign corporations with U.S. subsidiaries," review the results, and then refine the query: "Focus on cases after 2018 where the court found jurisdiction." On the finance side, a credit analyst evaluating a loan application might ask "Compare this borrower's debt-to-EBITDA ratio against our portfolio average for manufacturing companies," review the comparison, and follow up with "Show me how it trends over the last three years." What makes chat powerful is exactly this kind of iterative refinement. Each message is simply another event, processed through the same loop as everything else, just with tighter expectations for response time.

But that expectation for quick responses creates real design constraints. When you are waiting for an answer, every second of silence feels like something has gone wrong. Systems built for interactive use need to acknowledge requests immediately, show progress as work unfolds, and deliver results incrementally when possible. This might mean streaming partial responses as they are generated, displaying indicators that show which databases the system is searching, or breaking complex research into visible steps. Without this feedback, a blank screen followed by a complete response thirty seconds later leaves you wondering whether the system is working, stuck, or crashed.

Email routing lets agentic systems handle work that arrives through channels people already use. A general counsel might forward a compliance question from a business unit to a monitored inbox, and the system extracts the question, searches relevant guidance, and drafts a response for review. A portfolio manager might forward a client inquiry about sector exposure to a research assistant inbox, receiving back a summary with current allocations and recent changes. The challenge with email is that the requests tend to be unstructured, often buried in forwarded threads with multiple topics and tangential context that the system must parse to find the actual question.

Collaboration platforms like Slack and Teams let agentic systems appear as team members in the workflow. A deal team negotiating an acquisition can @mention the system in their working channel to check whether a proposed indemnification cap falls within market norms, without anyone switching applications or breaking the flow of discussion. An investment committee preparing for a meeting can request a quick portfolio risk summary in their coordination channel. Users can also

invoke specific capabilities through commands like `/research "material adverse change clauses"`. The convenience comes with a security consideration: collaboration platforms log all messages, and channels often include people with different access levels, so systems need to be careful about what information they surface and where.

Voice interfaces are best suited for quick lookups when typing is impractical: checking a position size while walking between meetings, confirming a filing deadline while reviewing documents, or getting a quick summary of overnight market moves during a commute. The tradeoffs are significant, though. Transcription errors can mangle legal and financial terminology, and without a keyboard, verifying identity becomes harder. For anything beyond simple information retrieval, voice interfaces should require explicit confirmation before the system takes action.

These synchronous channels share a common advantage: if the system misunderstands your request, you can clarify immediately. Asynchronous channels like email and background automation do not offer that safety net. When you send a request and walk away, you will not discover a misunderstood instruction until you return to find the wrong output. This is why intent understanding (Section 3) and perception design (Section 4) matter even more for asynchronous workflows. The system must figure out what you actually want from a single message, gather the right information without guidance, and deliver results that match your expectations on the first attempt. And because no one is watching, asynchronous systems need robust logging and notification so that operators can understand what happened after the fact.

2.3 Scheduled Jobs: Time as Trigger

Unlike the channels we have discussed so far, some work does not wait for an external event or a human request. End-of-day reconciliation, monthly compliance reporting, quarterly reviews, and annual filings all happen on a calendar. For these recurring tasks, time itself becomes the trigger, and the system must initiate work without anyone asking.

Calendar-driven deadlines govern much of legal and financial practice. Litigators live by deadlines: answer the complaint within 21 days, file motions 30 days before hearings, respond to discovery within 30 days. Missing one can mean default judgment or sanctions. An agentic system can monitor litigation calendars, calculate deadlines while accounting for court holidays and local rules, and escalate as deadlines approach if work remains incomplete. A more sophisticated system might go further, retrieving the complaint a week before the answer is due, extracting the claims, generating a draft answer with standard defenses, and presenting it for attorney review with time to spare. Financial professionals face analogous deadline pressure: quarterly SEC filings, annual audits, covenant compliance certificates due to lenders, and tax submissions that carry real penalties for delay. The same scheduled-trigger architecture that reminds a litigator about an approaching discovery deadline can remind a fund administrator about an investor reporting obligation.

Periodic compliance checks run on a schedule even when nothing externally triggers a review.

Consider the investment compliance team at an asset manager: every night, the system checks each portfolio against its investment policy statement, looking for positions that exceed concentration limits, securities that fall outside permitted categories, or exposures that have drifted beyond acceptable ranges. Violations get flagged for review before markets open. Law firms face a parallel challenge with conflicts of interest. A conflicts system can retrieve new docket entries and engagement letters each day, extract party names, and check them against the firm's conflicts database. If the same company name appears on both sides of a matter, the system escalates immediately rather than waiting for someone to notice. These scheduled checks enable continuous monitoring that would be impractical to perform manually across thousands of client accounts or active matters.

End-of-day and end-of-period workflows are critical in both industries. In finance, end-of-day processing is a well-established ritual: when markets close, systems retrieve final prices, mark positions to market, reconcile trades against counterparty confirmations, calculate profit and loss, and generate the risk reports that will be waiting on desks the next morning. If any step fails or produces unexpected results, the system escalates rather than distributing incomplete data. Law firms have their own periodic workflows, though they tend to run on longer cycles. At month-end, a billing system might remind attorneys to enter unbilled time, generate draft invoices based on matter budgets and fee arrangements, and flag anomalies for partner review before bills go to clients. Quarterly, a matter management system might review open matters for staleness and prompt responsible attorneys to update status or close inactive files. The triggers are different, but the architectural pattern is the same: time fires the event, the system performs its work, and humans receive the output or the escalation.

2.4 Escalation Events: When Agents Reach Their Limits

The previous three channel types bring work into the system from outside: external feeds push events, humans make requests, and scheduled jobs fire on the clock. Escalation events are different. They originate inside the system itself, when an agent reaches a limit and cannot proceed autonomously. The system generates an event signaling that it needs human intervention, transferring control to a decision-maker at precisely the moment when human judgment matters most.

Four types of escalation triggers appear most frequently in practice.

Budget exhaustion occurs when the system approaches resource limits it has been given. These might be token consumption caps, iteration counts, time limits, or cost thresholds. Imagine a due diligence agent working through a data room with thousands of documents. If it burns through its budget analyzing the first hundred contracts without reaching the financial statements, it needs to stop and ask: should I continue with more resources, or should a human reprioritize what I review first? Without this kind of budget-aware escalation, the system either stops arbitrarily or runs up costs without limit.

Low confidence triggers escalation when uncertainty is too high for autonomous action. A research

agent might find conflicting authority on a legal question, with a recent circuit court decision that appears to contradict older Supreme Court precedent. Rather than picking one and hoping for the best, the system should surface the conflict and let an attorney decide how to handle it. Similarly, a credit analysis agent encountering a borrower with an unusual corporate structure might recognize that its standard analysis framework does not apply cleanly and escalate for human review rather than forcing the square peg into a round hole.

Approval requirements force escalation for actions that require explicit human authorization regardless of the system's confidence. Some actions are simply too consequential to delegate fully, even when the system is certain it knows the right answer. Filing court documents, sending communications to clients or counterparties, executing trades above a certain size, or making public disclosures all warrant human sign-off. The system may draft the motion, compose the email, or prepare the trade ticket, but a human must authorize the action itself.

Errors and anomalies require human intervention when something unexpected happens. Tools fail, data sources become unavailable, or results do not make sense. A due diligence agent that finds revenue figures in a company's 10-K that do not match the numbers in its earnings press release should not proceed with potentially inconsistent data. A portfolio monitoring agent that cannot reach the pricing service should escalate rather than working with stale prices. These situations require human judgment to determine whether to wait, retry, use alternative sources, or proceed with acknowledged limitations.

We return to escalation design in depth in Section 9, including how to calibrate when systems should ask for help versus proceed autonomously, and how to design escalation interfaces that give humans the context they need to make good decisions quickly. The organizational structures that support escalation—tiered response models, accountability pathways, and governance reporting—are addressed in *Governing Agents*.

2.5 Surfaces: How Users Experience Agentic Systems

Triggers and channels are architectural concepts—they answer questions about event routing, message delivery, and system integration. **Surfaces** are user experience concepts—they answer questions about interaction modality, workflow integration, and interface design. These are different disciplines asking different questions, even when they address the same system.

We include surfaces in this architectural chapter for two reasons. First, surface choice constrains architectural decisions: a system designed for conversational interaction needs different trigger handling than one designed for background automation. Second, architects and UX designers must communicate: when a UX designer specifies that users need real-time research dialogue, the architect must understand what that implies for channel selection and response latency.

Conversational surfaces deserve special attention because they sit at the intersection of architecture and UX. When a litigator types a research question into a chat window, the interface serves two roles

simultaneously: it is the pathway through which the human prompt trigger arrives (an architectural function) and the interaction modality through which the user engages with responses (a UX function). This dual role is unique to conversational surfaces—automation and document surfaces do not receive triggers, they only deliver outputs.

Surface design warrants extended treatment in its own right. Questions of conversational UX, interface affordances, and workflow integration deserve the depth we give them elsewhere in this book. Here, we focus on the essentials that architects need to understand: what surfaces exist, how they differ, and how surface choice shapes system requirements.

Surface choice matters because it shapes workflow integration. An agentic system with identical capabilities will succeed or fail based partly on whether its surface matches how professionals actually work. Usability researcher Jakob Nielsen argues that generative AI represents the first new user interface paradigm in sixty years, marking a shift from *command-based interaction*, where you tell the computer what to do, to *intent-based outcome specification*, where you tell the computer what you want (Nielsen 2023). But intent-based systems still require interfaces, and those interfaces—the surfaces through which users engage—determine whether the system feels natural or intrusive.

Interaction Surfaces

An **interaction surface** is the user experience modality through which humans engage with an agentic system. Where triggers and channels are architectural concepts describing how work reaches the system, surfaces are UX concepts describing how users experience the system. Surfaces vary along four dimensions that UX designers use to match interaction paradigms to user needs: synchronicity, initiative, embodiment, and output format.

Synchronicity determines when results arrive. Synchronous interactions deliver results while the user waits, as with chat interfaces where you ask a question and watch the response stream in. This suits exploratory work where you refine direction based on what you see. Asynchronous interactions deliver results later through notifications or reports, as with document generation where you specify requirements, do other work, and return when the draft is ready. This suits production tasks where waiting would waste billable time.

Initiative captures who starts the conversation. In pull models, the system waits for the user to initiate. A research assistant that answers questions when asked operates this way, giving the user control over when and whether to engage. In push models, the system reaches out when something needs attention. A docket monitor that alerts you to new filings operates this way, deciding when to interrupt based on relevance and urgency.

Embodiment refers to whether the system is visible in the workflow. Visible systems appear as explicit participants, like a chat avatar or copilot sidebar that makes the agentic system's presence clear. Users know they are interacting with AI and can direct it consciously. Invisible systems

operate as infrastructure, like a compliance screening system that flags suspicious transactions in the background. Users experience the outputs without seeing the system that produced them.

Output format determines what the system produces. Conversation turns suit iterative exploration where direction emerges through dialogue, producing chat transcripts from research queries, brainstorming, and strategy discussions. Structured documents suit formal deliverables with defined formats, producing research memos, due diligence reports, and contract summaries ready for distribution. Actions in the world suit automation workflows, where filing a document, sending an alert, or executing a trade produces effects rather than text.

Three primary surfaces dominate current deployments, each suited to different task types and user contexts.

Conversational surfaces present the agentic system as an interactive dialogue partner, best suited for exploratory tasks where users refine direction through iteration—researching unfamiliar areas, exploring scenarios, thinking through strategy. The interaction is synchronous and user-initiated. Limitations include the "articulation barrier" (users must express intent in natural language) and lack of visual **affordances** (no menus to browse or buttons to discover capabilities).

Automation surfaces present the agentic system as invisible infrastructure that monitors, analyzes, and acts in the background. Users receive outputs only when something relevant happens. Examples include portfolio surveillance that alerts when positions breach limits, docket monitoring that flags new filings in active matters, and compliance systems that screen transactions against sanctions lists. The interaction is asynchronous and system-initiated.

Automation surfaces suit *monitoring tasks* where continuous human attention is impractical. A compliance officer cannot manually review every transaction, and a litigator cannot check every docket daily. The agentic system handles the routine cases, surfacing only exceptions that require human judgment. The user experience is defined by what *does not* happen, since no alert means no problem.

Document surfaces present the agentic system as a drafting assistant that produces structured work products such as research memos, due diligence reports, deposition summaries, and client presentations. The user specifies requirements, the system produces a document, and the user reviews, edits, and distributes. The interaction is asynchronous because the system works while the user does other things, but it remains user-initiated.

Document surfaces suit *production tasks* with defined deliverables. The associate needs a memo for the partner's review, and the analyst needs a report for the investment committee. The output format matters here because you need a polished document that can be filed, sent to clients, or presented to regulators, not a chat transcript.

Matching Surface to Task

Surface selection is a design decision, not a technical constraint. When choosing how users will interact with an agentic system, match the surface to the task type.

Exploratory tasks like research questions, strategy discussions, and ad hoc analysis work best through **chat surfaces** that support iterative refinement.

Monitoring tasks like docket surveillance, compliance screening, and portfolio alerts work best through **automation surfaces** that operate in the background.

Production tasks like research memos, due diligence reports, and regulatory filings work best through **document surfaces** that generate formal deliverables.

Many deployments combine all three. A litigation support system might offer chat for research, automation for alerts, and document generation for motion drafts. The underlying reasoning capabilities remain constant while only the interaction modality changes.

Mismatches waste effort. Forcing chat onto monitoring tasks demands attention no one can sustain. Forcing documents onto exploratory tasks prevents the iteration that produces good results.

The interplay between triggers, channels, and surfaces creates the complete human-agent interface. Table 2 illustrates how the same trigger arriving through the same channel can manifest through different surfaces depending on workflow context.

Table 2: How triggers, channels, and surfaces combine in practice

Trigger	Channel	Surface	Workflow
New court filing	External feed	Automation	Background monitoring; alert when needed
New court filing	External feed	Chat	Interactive research on opposing brief
New court filing	External feed	Document	Case summary memo for partner
Price threshold crossed	External feed	Automation	Background alert to portfolio manager
Price threshold crossed	External feed	Chat	Interactive rebalancing analysis
Price threshold crossed	External feed	Document	Trade recommendation for committee
Research question	Human prompt	Chat	Iterative legal research
Report request	Human prompt	Document	Due diligence memo

Emerging surfaces include embedded copilots (agentic capabilities integrated into existing applications via sidebars or inline suggestions), ambient interfaces (voice-based interaction, though currently limited by transcription errors in professional contexts), and agentic APIs (machine-to-machine orchestration where the "user" is another system—see Section 10).

Surface design draws on decades of human-computer interaction research. What architects need to understand is that surface selection has architectural consequences: conversational surfaces require low-latency response paths, automation surfaces require robust background processing, and document surfaces require generation pipelines. UX designers choose surfaces based on user needs; architects build the infrastructure those surfaces require.

2.6 Evaluating Trigger Systems

When evaluating trigger systems, assess five criteria: *coverage* (all relevant event sources), *latency* (acceptable delay for the use case), *reliability* (failure handling and fallbacks), *priority* (distinguishing urgent from routine), and *auditability* (complete trigger logs). See the consolidated Evaluation Checklist in **??**.

2.7 From Triggers to Action

This section has examined the human-agent interface from two disciplinary perspectives. From an architectural standpoint, *triggers* are the events that initiate work, and *channels* are the pathways through which those triggers reach the agent. From a user experience standpoint, *surfaces* are the interaction modalities through which users engage with the agent's capabilities. Architects design trigger routing and channel infrastructure; UX designers choose surfaces that match how professionals actually work. The disciplines address different questions but must align for a system to succeed.

Together, triggers, channels, and surfaces determine how an agentic system fits into professional workflows. A well-designed trigger system ensures no relevant event goes unnoticed. Appropriate channel selection balances latency against reliability. Thoughtful surface choices match interaction modality to task type. And all three must support the audit trails that governance requires—every action traceable to the trigger that initiated it, through the channel that delivered it, via the surface through which it was experienced.

But triggering is only the beginning. Once an event arrives, the agentic system must:

- **Understand intent**: What is being asked?
- **Perceive information**: What does the system need to know?
- **Take action**: What should the system do?
- **Remember context**: What should persist across sessions?
- **Plan execution**: How should work be decomposed?
- **Recognize completion**: When is the task done?
- **Escalate appropriately**: When should humans intervene?

To see how these questions connect, trace a single event through the system. An external feed delivers a court filing notification, and the system routes it based on matter type and urgency. Depending on the configured surface, a litigator might receive a chat prompt inviting interactive research, a

background alert waiting in their queue, or a generated memo summarizing the filing's implications. Whatever the surface, the system retrieves case context from memory, downloads the filed document, analyzes its content, and either proceeds autonomously or escalates for human judgment. Each step in this chain—from trigger to channel to surface to action—represents an architectural decision with consequences for what the system can accomplish and how reliably it performs.

With work arriving through triggers and channels, and users engaging through surfaces, the next question becomes: how does the agentic system understand what is actually being asked? Section 3 takes up this problem of intent.

3 How Does an Agent Understand What's Being Asked?

When a partner walks into your office and says "get me up to speed on the Acme acquisition," your first job is understanding what that actually means. You must determine whether this is a quick status check or a request for deep analysis, whether you should answer a specific question or identify all issues, and whether this is urgent work for today's call or background work for next week's meeting. The words you hear are the **instruction**; the underlying purpose that those words point toward is the **intent**.

Every professional develops this skill over time: reading the assignment memo, clarifying ambiguous instructions, and understanding not just what was said but what was meant. Junior associates tend to over-clarify, whereas senior associates internalize firm norms and client expectations and infer appropriately. The best professionals know when to ask and when to proceed.

Agentic systems face the same challenge. The user provides an instruction in natural language, which is inherently ambiguous and sometimes contradictory. Despite this, the agent must extract the user's intent to determine what goal is being pursued, what constraints apply, and what success looks like. This illustrates the second fundamental question: *How does an agent understand what's being asked?*

Four concepts structure this process: instruction, intent, goal, and task. Understanding how they relate—and where gaps arise—is essential for building and governing agentic systems.

Instruction

An **instruction** is the words the user provides—the raw input that starts the process.
"Review this credit agreement for risks."
"Rebalance to reduce tech exposure."
"Get me up to speed on the Acme acquisition."

Instructions are where work begins, but they are rarely complete specifications. The word "risks" raises immediate questions: risks to whom? The lender or borrower? Material risks or all risks? Legal

risks, financial risks, or both? "Reduce tech exposure" leaves open the target level, the mechanism (sales, hedges, or both), and tax and timing constraints. "Get me up to speed" specifies neither depth, urgency, nor deliverable format. Instructions are clear enough to start; they are not clear enough to finish.

Pre-LLM systems handled instructions through rigid steps: matching exact words, filling forms, simple if-then checklists. These systems worked for narrow domains with controlled vocabularies but broke on natural language variation. "Find cases on personal jurisdiction" and "What's the law on where you can sue someone?" express similar meanings but look nothing alike to a keyword matcher.

Intent

Intent is the underlying purpose, constraints, and success criteria behind an instruction. Intent captures the human meaning—what the user actually wants, not just what they said. Where an instruction might be "review this credit agreement," the intent might be "identify material risks to the lender by tomorrow for the partner's client call."

This framing assumes the user has clear intent that merely needs to be extracted. Often, they do not. A general counsel asking for "a quick overview of the regulatory landscape" may not understand the domain well enough to know what a useful overview would contain or how comprehensive it needs to be. A partner requesting "a one-page memo on the indemnification issues" may be asking for something impossible if the issues genuinely require five pages to explain competently. A portfolio manager who wants "maximum returns with minimal risk" is expressing a preference, not a coherent goal. Users operate under their own constraints—time pressure, cognitive load, incomplete domain knowledge—that shape requests in ways that may conflict with the realities of the underlying work. Intent inference must contend not only with unclear expression but with intent that is itself incomplete, conflicted, or impossible to satisfy.

The gap between instruction and intent has always existed; what has changed is our ability to bridge that gap with technological systems. Early AI research on dialogue established that understanding utterances requires inferring the speaker's underlying plans and goals (Allen and Perrault 1980). Large language models dramatically improved intent inference from natural language, particularly after techniques like reinforcement learning from human feedback (RLHF) explicitly optimized models to follow user intent (Ouyang et al. 2022). Where rule-based systems required exact matches, LLMs handle variation, implicit context, and domain-specific jargon. Modern LLMs excel at handling noisy input (misspellings, shorthand, tangential information), resolving references using conversational context, inferring domain-specific meaning, and detecting implicit constraints that professionals take for granted.

Despite these capabilities, intent understanding remains imperfect. As conversations extend, LLMs

may lose track of earlier context or constraints. When clarification is needed, LLMs sometimes proceed with a default interpretation rather than asking, resulting in a "helpful but wrong" failure: the agent does *something* reasonable rather than confirming it understood correctly. Professionals also communicate through implication. Phrases like "This needs to be right" signal high stakes, while "When you get a chance" signals low urgency. These signals may not be explicitly parsed by current models.

Goal

A **goal** is the machine-readable specification that intent points toward—the structured outcome that would satisfy the request. Goals connect to the GPA+IAT framework introduced in the previous chapter: an agent pursues goals through perception and action. Where intent is "identify material risks to the lender by tomorrow," the goal is "produce a lender-risk memo that meets firm policy and is ready for partner review."

Goals provide the target for planning and the criterion for termination. An agent that understands the goal can determine whether its work is complete: does this memo identify the material risks? Does it meet firm standards? Is it ready for the partner? Without a clear goal, the agent cannot know when to stop—the "runaway associate" problem of endless research without a deliverable.

Task

A **task** is the concrete unit of work the agent executes to advance a goal. A single goal typically decomposes into multiple tasks. For the lender-risk memo, tasks might include extracting financial covenants, comparing covenant terms to market standards, identifying collateral coverage gaps, flagging cross-default provisions, and drafting the summary memo.

Tasks are where planning meets execution. Section 7 addresses how agents decompose goals into task sequences; Section 8 addresses how agents recognize when tasks—and goals—are complete. Both capabilities depend on clear intent: ambiguous goals yield uncertain plans and unreliable termination criteria.

The flow from instruction to task is: **Instruction → Intent → Goal → Tasks**. Inferring intent bridges the gap between what users say and what they mean. Goals give agents targets to aim for. Tasks give agents concrete work to execute. Failures at any stage propagate forward: misunderstood instructions yield wrong intent; wrong intent yields wrong goals; wrong goals yield wasted tasks.

3.1 Bridging the Gap

Traditional enterprise systems classified work using explicit **routing rules**: fixed logic that checked message details and forwarded them to the right handler. A court filing tagged `matter_id=12345` goes straight to litigation monitoring; a portfolio company's SEC filing routes to compliance re-

view. This pattern dominated middleware architecture for decades: message queues, **enterprise service buses (ESBs)**, and workflow engines functioned like a mailroom sorting notices to the right department.

LLM-based agentic systems replace explicit routing logic with semantic reasoning. Rather than maintaining explicit rules for every event type, the model *understands* what kind of work this is. "Review this credit agreement" and "Check this loan doc for problems" express similar intents despite different words; the LLM recognizes both as document review tasks without explicit rules mapping each phrase to a handler.

This architectural shift creates two primary tensions in system design.

The first tension is between flexibility and predictability. Rule-based routing is deterministic: the same input always produces the same classification. LLM-based classification relies on likelihoods and may shift slightly with model updates, prompt phrasing, or context. For regulated applications, this requires additional governance through logging classifications, monitoring for drift, and maintaining override rules for critical categories.

The second tension involves explicit versus implicit knowledge. Routing rules encode domain knowledge explicitly in code, requiring developers to anticipate every category and write rules to match. LLM classification absorbs domain knowledge implicitly through training, recognizing that legal research and case analysis are related without explicit rules. But implicit knowledge is harder to audit because there is no specific rule to inspect, and it may reflect training biases.

Production systems often combine both patterns. Simple, high-volume, time-sensitive classifications use deterministic rules, so a margin call always routes to the trading desk. Complex, ambiguous, or novel requests use LLM reasoning. The rule-based layer handles predictable cases efficiently while the LLM layer handles everything else.

For architects evaluating agentic systems: where must classification be deterministic (regulatory requirements, latency/speed constraints, auditability)? Where does flexibility justify the overhead of LLM reasoning? The answer shapes system design.

Intent Inference Is Not Mind Reading

LLMs infer *probable* intent from language patterns; they do not read minds. Inference fails when:

- The instruction is genuinely ambiguous (multiple reasonable interpretations)
- The user's intent differs from typical patterns for similar language
- Critical context exists outside the conversation (prior meetings, firm norms)
- The user themselves is unclear about what they want

Design for clarification, not guessing. The aim is an agent that surfaces uncertainty and asks, not one that pretends it possesses certainty regarding intent.

3.2 Goal Extraction from Natural Language

Once the agent receives an instruction, it must extract structured goals that can guide execution. This extraction transforms natural language into actionable specifications through two main processes: intent classification and constraint recognition.

The first step, intent classification, translates the instruction into task types that determine workflow. This step converts raw words into candidate tasks that can advance the underlying goal:

- **Information retrieval**: "What's the current NAV (Net Asset Value)?" "Find the latest 10-K"
- **Research and analysis**: "Research whether we can pierce the corporate veil (hold shareholders liable)"
- **Document review**: "Review the acquisition agreement for change-of-control provisions"
- **Document generation**: "Draft an engagement letter for the Smith matter"
- **Calculation**: "Calculate the IRR (Internal Rate of Return) assuming a 5-year hold"
- **Monitoring**: "Alert me if tech exposure exceeds 30%"

Different task types invoke different tools, planning patterns, and success criteria. A research task requires search and synthesis; a calculation task requires structured computation; a monitoring task requires continuous observation.

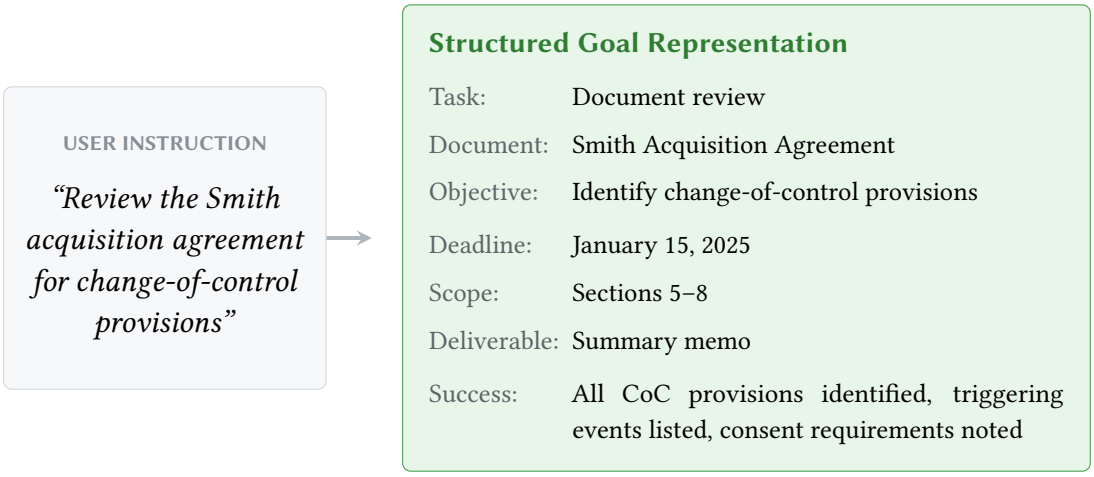
Beyond classification, the agent must also recognize entities and constraints. Entity recognition identifies what the task concerns: matters, clients, securities, parties, documents, and jurisdictions. When someone says "Review the Smith acquisition agreement," the agent must recognize a reference to a specific document; "Research Delaware fiduciary duties" references a jurisdiction that shapes which law applies.

Constraint recognition identifies what bounds apply to execution. Temporal constraints include deadlines, as-of dates, and time windows. "By Friday" sets a deadline; "as of year-end 2024" sets a reference date; "over the past quarter" defines a window for analysis. Resource constraints set budget and effort limits, with phrases like "Spend no more than 2 hours" or "focus on Articles 3 and 4" bounding scope. Format constraints specify how deliverables should appear: "Summarize in one page" constrains length, "prepare a memo for the file" specifies format, and "I need something to show the client" signals an external audience requiring different tone and detail.

Two additional constraint types often remain unstated. Audience and privilege constraints determine who will see the output and what confidentiality must be preserved. Risk and compliance constraints set limits that professionals internalize but rarely articulate: a compliance review implicitly requires flagging violations, and a client communication implicitly requires privilege protection.

Once extracted, these components can be organized into structured goal representations that guide

execution. These representations resemble short assignment memos that the planning system can act on (Section 7) and the termination system can measure against (Section 8).



3.3 Ambiguity Detection and Clarification

Not all instructions can be unambiguously interpreted. The agent must detect ambiguity and decide whether to clarify or proceed. The decision depends on two factors: ambiguity severity and action stakes.

When both stakes and ambiguity are low, the agent should proceed with its best interpretation. If someone asks "What's Apple's market cap?" and there is slight uncertainty about whether they mean Apple Inc. or Apple Hospitality REIT, the dominant interpretation is obvious and the cost of being wrong is low since correction is easy. When stakes remain low but ambiguity is high, a brief clarification prevents wasted effort. If someone asks "Research the statute of limitations" without specifying the claim type, a quick question saves hours of potentially misdirected work.

When stakes are high but ambiguity is low, the agent should confirm before acting. If the instruction is clear but consequential, such as "File this motion," confirmation prevents irreversible errors even when the agent is confident it understood correctly. When both stakes and ambiguity are high, thorough clarification is essential. If someone says "Handle the regulatory response" for a complex matter, extended clarification is appropriate before taking any action.

Effective clarification has four characteristics. It is specific, asking "Which jurisdiction's statute of limitations: Delaware or New York?" rather than "Can you clarify?" It is contextual, referencing what the agent already understands: "I understand you want me to review the credit agreement. Should I focus on lender protections, borrower obligations, or both?" It is actionable, offering options rather than open-ended questions: "Should I (a) provide a comprehensive review of all provisions, (b) focus on the financial covenants, or (c) flag only provisions that differ from our standard template?" And it is bounded, limiting clarification rounds. If the agent needs extensive clarification, it may be the wrong tool for the task, or the user may need to think through requirements before delegating.

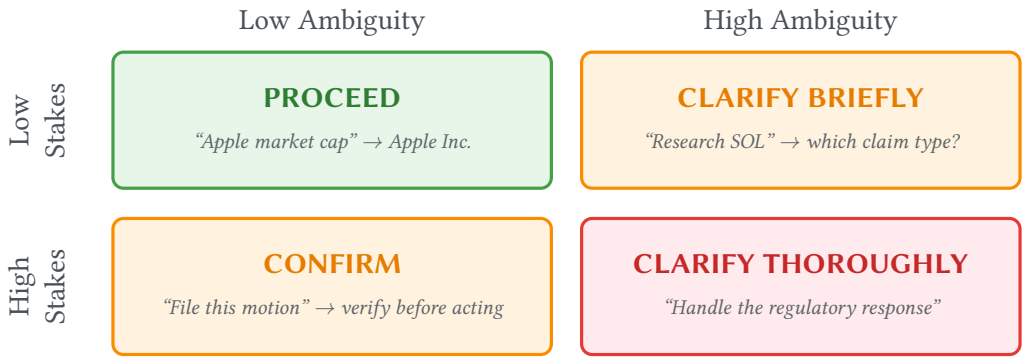


Figure 2: Decision matrix for when agents should clarify user intent. Stakes measure consequence of error; ambiguity measures interpretation confidence.

> *Poor clarification*: "Can you clarify?"
> *Better*: "Should I assess lender risks, borrower risks, or both?"
> *Best*: "You asked to reduce tech exposure. Should I (a) sell tech to 25% target, (b) hedge with options, or (c) add non-tech positions? Which deadline matters—this week or month-end reporting?"

Research has documented that LLMs sometimes select a default interpretation rather than asking for clarification, even when ambiguity is significant (Zhang et al. 2024; Wang et al. 2024b). This "proceed without asking" behavior creates real risk: the agent interprets "review the contract" as a surface-level summary when the user expected deep issue-spotting, delivering work product that is technically responsive but wrong.

Several strategies can mitigate this tendency: prompt engineering that emphasizes clarification for ambiguous requests, confidence thresholds that trigger clarification below a certainty level, user training to provide detailed initial instructions, and checkpoint reviews before significant work begins. From a governance perspective, teams should monitor for cases where the agent proceeded confidently but delivered unexpected results. These cases may indicate calibration problems in ambiguity detection.

Ambiguity detection assumes the user has clear intent that was merely expressed unclearly. Sometimes the problem runs deeper.

When Intent Itself Is Unclear or Conflicted

Domain expertise gaps: The user may not understand the subject well enough to specify what they need. "Give me the key issues" presumes the user knows what issues exist and which matter most—knowledge they may be asking the agent to provide.

Conflicted constraints: The user may want contradictory things. A one-page memo on a complex acquisition may be impossible without sacrificing the accuracy or completeness that makes the memo useful.

Unexamined tradeoffs: The user may not have considered tradeoffs they would care about if surfaced—speed versus thoroughness, cost versus quality, brevity versus nuance.

When an agent detects these deeper problems, it should surface the conflict rather than silently resolving it: "You asked for a one-page summary, but covering the indemnification, representations, and covenant issues adequately would require three to four pages. Would you prefer (a) a one-page overview that flags issues without detailed analysis, (b) a longer memo with full analysis, or (c) detailed treatment of just one area?" This is harder than detecting ambiguous expression because it requires recognizing when the user's own mental model is incomplete or conflicted—and surfacing that recognition without appearing to second-guess or condescend.

3.4 Constraint Identification

Beyond explicit instructions, agents must identify constraints that bound acceptable execution. These constraints fall into several categories that often interact.

Temporal constraints include deadlines and time windows. Some are explicit, like "by Friday." Others are implicit, such as court filing deadlines calculated from procedural rules. Still others are contextual: "before the board meeting" requires knowing when the meeting is scheduled. Resource constraints set budget and effort limits. Token budgets limit API costs; time budgets limit calendar impact; scope constraints focus effort on high-value areas.

Scope constraints define what is in and out of bounds. "Focus on Articles 3 and 4" excludes other articles; "just the Delaware analysis" excludes other jurisdictions. Format and style constraints specify how deliverables should appear: memo versus email versus presentation, formal versus casual tone, internal versus client-facing audience. Risk and compliance constraints specify what must be avoided: privilege protection, conflicts of interest, regulatory restrictions, and confidentiality obligations. These constraints often apply implicitly based on context.

Professionals operate under many constraints they rarely state explicitly. When a partner says "research Section 10(b) liability," implicit constraints include:

- Use authoritative sources (binding precedent, not blog posts)
- Focus on the relevant jurisdiction (probably the circuit where the case is filed)

- Assume current law (not historical analysis unless specified)
- Protect privilege (don't disclose strategy in external searches)
- Operate within budget norms (don't spend 40 hours on a 2-hour task)

Agents must infer these constraints from context, domain knowledge, and organizational norms. Memory systems (Section 6) are essential here: they preserve firm-specific expectations across tasks, user profiles accumulate individual preferences, and matter context provides case-specific constraints that sharpen future intent interpretations.

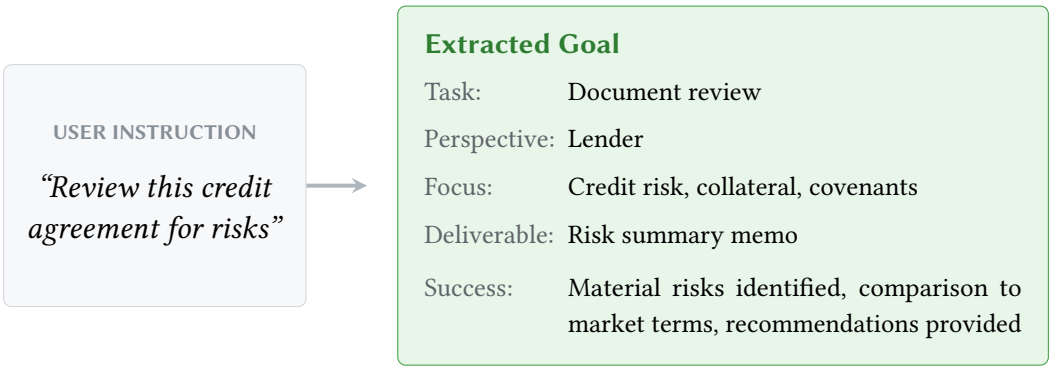
3.5 Validation and Domain Examples

Before executing, agents should validate their understanding of intent. Several patterns support this validation.

Reflection and summarization involve the agent pausing—much like an associate double-checking their notes—to restate its understanding before proceeding, giving the user an opportunity to correct misunderstandings before work begins. **Chunked validation**, like partner check-ins after each memo section, breaks complex tasks into phases rather than validating all at once. After completing research, the agent summarizes findings and confirms direction before drafting. After drafting, it confirms the approach before finalizing. Each checkpoint prevents error propagation.

Confidence signaling requires the agent to indicate how confident it is in its own understanding. When confidence is high, the agent can proceed with light oversight; when confidence is low, the right move is to pause and ask for clarification rather than press ahead. Clear confidence signaling helps users decide how much review is needed and whether to treat the output as a draft, a starting point, or a near-final product.

Consider how intent extraction and validation work together for a legal task. Given the instruction "Review this credit agreement for risks," the agent classifies this as a document review task and detects that "risks" is ambiguous (risks to whom? what types?). Context gathering reveals this is a lender-side engagement for a senior secured facility. The agent infers implicit constraints (focus on lender risks, prioritize material issues, assume current market terms as baseline) and clarifies: "I'll review from the lender's perspective, focusing on credit risk, collateral coverage, and covenant adequacy. Should I also flag documentation risks (drafting issues, missing provisions) or focus only on substantive credit terms?"



The agent validates: "I'll prepare a memo identifying material risks to the lender, comparing key terms to market standards, and recommending negotiation points. I'll have a draft for your review by tomorrow afternoon."

The same pattern applies to financial tasks. Given the instruction "Rebalance to reduce tech exposure," the agent classifies this as a portfolio action task and immediately detects multiple ambiguities: how much reduction? through what mechanism? with what constraints? Context gathering reveals current tech exposure at 35% against a 25% target. Through clarification dialogue, the agent confirms the user wants to reach target through sales while minimizing tax impact (preferring loss harvesting—selling assets at a loss to offset gains—and long-term gains over short-term).

Intent Understanding Is Continuous

Intent extraction is not a one-time step at task initiation. As the agent works, it may discover:

- The original understanding was incomplete (new constraints emerge)
- The user's intent has evolved (priorities shift mid-task)
- Implicit constraints conflict (cannot optimize for both)
- The task is impossible as specified (constraints are mutually exclusive)

Effective agents surface these discoveries through clarification rather than proceeding with outdated or impossible goals. Intent understanding is iterative, not instantaneous.

Intent understanding connects to other framework questions. Memory (Section 6) improves intent extraction over time by preserving user preferences, matter history, and firm norms. Planning (Section 7) depends on clear intent; extracted goals feed the planning system, while ambiguous intent propagates through the plan as uncertainty. Governance must address intent misalignment as a core risk, verifying goal alignment before deployment and monitoring for drift during operation. *Governing Agents* examines these controls through the lens of goal dynamics calibration, distinguishing static, adaptive, and negotiated goals and specifying appropriate oversight for each.

Understanding intent bridges the gap between what users say (instruction) and what they mean (intent), shaping the goals and tasks the agent will plan. Clarification beats guessing when ambiguity

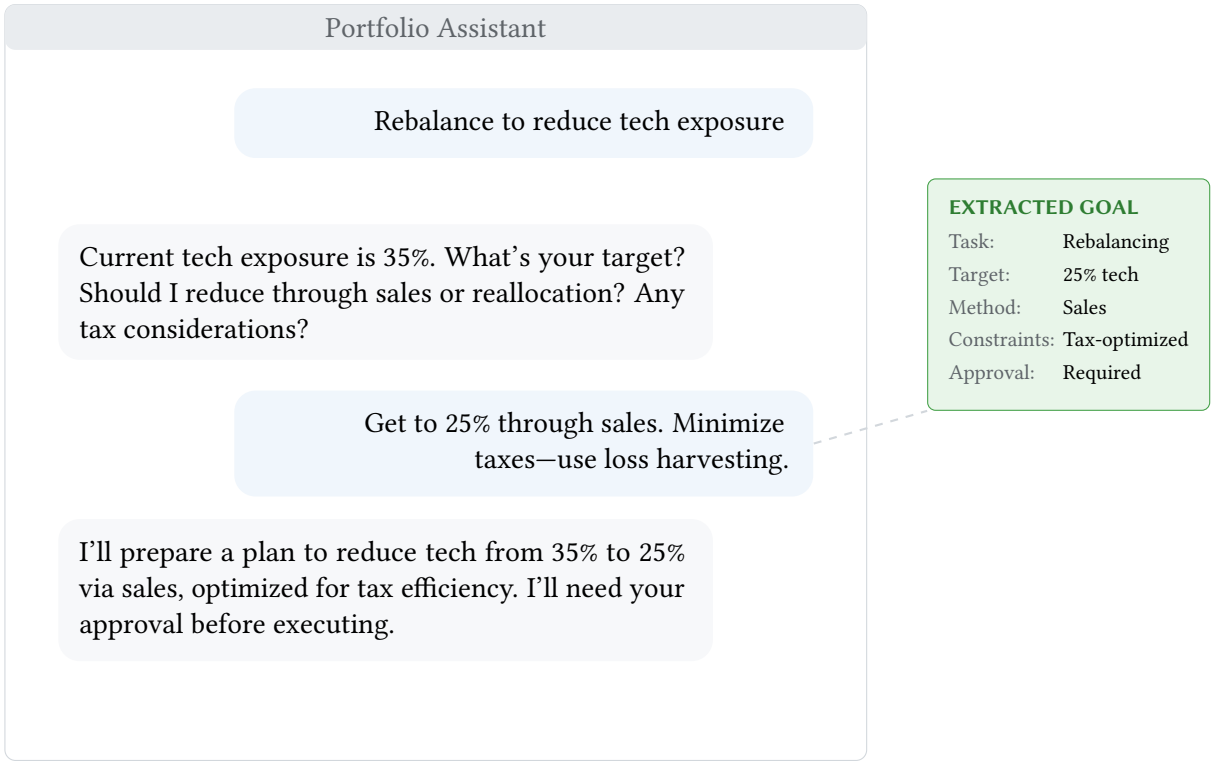


Figure 3: Clarification dialogue for a portfolio rebalancing task. The agent detects ambiguity, gathers constraints through targeted questions, and extracts a validated goal before proceeding.

is significant and stakes are high. Constraints—time, scope, audience, compliance, and budget—matter as much as goals. Validation prevents wasted effort by confirming understanding before significant work begins.

With triggers delivering work and intent extraction revealing what's being asked, the agent faces a practical problem: extracted goals require information the agent does not yet have. The credit agreement analysis task requires the actual credit agreement. The research question requires access to case law databases. The rebalancing plan requires current portfolio positions and market prices. Understanding what you need to do is not the same as having what you need to do it.

Section 4 examines the next question: how does an agent find things out? Perception tools—the interfaces to external information sources—bridge the gap between understanding a task and executing it.

4 How Does an Agent Find Things Out?

Understanding what someone wants is not the same as being able to deliver it. The previous section examined how agents interpret instructions, extracting goals, detecting ambiguity, and gathering the context needed to proceed. But even an agent that perfectly understands "Research Ninth Circuit authority on personal jurisdiction for foreign corporations" cannot help if it lacks access to case law databases. An agent that correctly interprets "Analyze this credit agreement from the lender's perspective" is useless without the credit agreement itself.

This constraint will feel familiar to anyone who has onboarded a new professional. A junior associate's effectiveness depends as much on access as on reasoning ability. Can they query Westlaw? Do they have credentials for the Bloomberg terminal? Can they search the firm's precedent database and document management system? The answers to these questions determine which problems they can solve. A brilliant analyst without access to portfolio data reasons in a vacuum; a talented associate without access to the case file works blind.

Agentic systems face exactly the same constraint. A large language model can reason impressively about legal doctrines and financial concepts, but without integrations into external or internal sources, it cannot access current case law, live market prices, client documents, or regulatory filings. These connections are what we call *perception tools*: the interfaces through which an agent observes the world beyond its training data.

Tools and Perception

A **tool** is a function that allows an agent to interact with external systems. **Perception tools** are the subset that are read-only: they let the agent observe without changing anything. When an agent queries a database, retrieves a document, or fetches market data, the external system's state remains unchanged.

Perception implements the "P" in the GPA+IAT framework from Part I. It defines the boundary of what information the agent can access, and that boundary matters enormously. An agent with access to public filings reasons differently than one with access to internal deal documents. An agent that can query real-time market data operates differently than one limited to end-of-day prices.

This section examines perception: the read-only tools that let agents gather the information they need. The next section, on action, examines write tools that let agents change things in the world. The distinction matters for governance: reading a document and sending an email carry fundamentally different risks, and your controls should reflect that difference. Section 5 develops these distinctions and the different oversight mechanisms each requires.

4.1 Perception Tool Categories

Not all perception tools work the same way, and choosing the right tool for the task matters. Perception tools generally fall into three categories: information retrieval, document processing, and computation.

Information retrieval tools query external platforms and databases to bring back answers. On the legal side, these tools connect to research services like Westlaw and Lexis in the United States, Beck-Online in Germany, LawNet in Singapore, and similar platforms in other jurisdictions, allowing agents to search case law, retrieve full opinions, check citing references, and download court filings. On the finance side, similar tools connect to platforms like Bloomberg, FactSet, or Refinitiv, enabling agents to pull real-time prices, retrieve company fundamentals, and access analyst research. Many organizations also have internal knowledge bases, including document management systems, deal archives, and precedent databases, that agents need to search to find prior work product relevant to a current matter.

Document processing tools transform raw files into data an agent can actually work with. A credit agreement arrives as a PDF; a financial statement comes as a scanned image; a data room contains thousands of files in mixed formats. Before an agent can reason about this content, it needs to extract the text (using OCR for scanned documents), identify what type of document each file is, and pull out structured information like party names, dates, and dollar amounts. During due diligence, for example, an agent reviewing a data room must distinguish contracts from correspondence, extract key terms from each agreement, and organize findings in a way that supports analysis. Without

document processing tools, the agent sees only filenames and metadata.

Computation tools generate new information through calculation rather than lookup. Consider deadline calculation: the Federal Rules of Civil Procedure require an answer within 21 days of service (Legal Information Institute 2024), but determining the actual due date requires accounting for weekends, court holidays, and local rules. That is a computational task, not a database query. Citation formatters perform a similar function, converting case information into proper Bluebook or other citation styles. In finance, computation tools normalize security identifiers (mapping between tickers, CUSIPs, and ISINs), calculate risk metrics like Value at Risk from position data, or derive analytics that inform investment decisions. These tools produce new information for the agent to use without changing anything in the external world.

4.2 In-Context Learning and Retrieval

Before examining specific retrieval mechanisms, we must understand the fundamental capability that makes retrieval useful: **in-context learning**. This concept is central to how modern language models work and why retrieval matters.

In-Context Learning

In-context learning (ICL) is the ability of large language models to learn from information provided in their input—without any update to the model's underlying parameters (Brown et al. 2020). When you provide examples, documents, or instructions in a prompt, the model adapts its responses based on that context. This is not "learning" in the traditional machine learning sense of updating weights; it is learning *within the conversation.*

In-context learning is what makes retrieval valuable. When an agent retrieves a relevant statute or prior memo and includes it in its prompt, the model can reason about that specific content even though it never saw it during training. The model's behavior changes based on what appears in its context window.

In-context learning explains why agents can work with information far newer than their training data. A model trained in 2024 can analyze a regulation enacted in 2025—provided that regulation appears in its context. The limitation is the **context window**: the maximum amount of text the model can process at once. Current models handle roughly 200,000 to 1,000,000 tokens (where a token is roughly three-quarters of a word),[1] but the cost, speed, and quality of results vary widely—and dangerously—when operating on large amounts of input. A model may technically accept a million tokens while producing degraded results on tasks requiring attention to details scattered throughout. Professional knowledge bases contain millions of documents regardless; the gap between what fits in context and what exists in the world creates the need for retrieval.

[1] Context window sizes as of December 2025 (vendor-reported): Claude Opus 4.5 (200k tokens), Gemini 3 (up to 1M tokens), GPT-5.2 (256k tokens). These specifications change frequently; consult current model documentation.

Why Agentic Architecture Works

The practical limits of context windows explain much of the value of agentic systems. Rather than attempting to process an entire knowledge base in a single prompt—which would exceed limits and degrade quality regardless—an agent decomposes work into focused steps, retrieves only what is relevant to each step, and reasons over smaller, manageable contexts.

This is not merely automation; it is an architectural solution to a fundamental constraint. Planning (Section 7) breaks complex tasks into subtasks, ensuring each retrieval step is targeted rather than overwhelming. Retrieval fetches specific information for each subtask. The LLM operates within its effective range on each step, even when the overall task would be impossible to handle monolithically.

This decomposition explains both why agentic systems are powerful and why they fail in predictable ways. The reliability cliff (Section 8) shows that short, focused tasks succeed while long tasks fail—precisely because decomposition keeps individual steps within the model's effective operating range, but errors compound across many steps.

Retrieval Methods. To retrieve relevant information from large collections, we need a way to find what matters. Several approaches exist, each with distinct strengths:

Keyword and boolean search is the oldest and most familiar approach. Platforms like Westlaw, Lexis, and Bloomberg have offered sophisticated keyword search for decades. Boolean operators (AND, OR, NOT), proximity searches, and field-specific queries give users precise control. When you need documents containing exact terms—a specific case citation, a statutory section, a company name—keyword search excels. Its limitation is vocabulary mismatch: a search for "breach of fiduciary duty" will not find documents discussing "violation of trust obligations" unless both phrases happen to appear.

BM25 and probabilistic search extends keyword matching with statistical weighting (Robertson and Zaragoza 2009). Terms that appear rarely in the corpus but frequently in a document signal relevance more strongly than common terms. BM25 powers many production search systems, including Elasticsearch and the baseline retrievers in academic benchmarks. It requires no machine learning infrastructure—just an inverted index—making it fast, interpretable, and cheap to operate.

Embedding-based semantic search uses machine learning to encode meaning numerically. An **embedding** is a vector—a list of numbers—that represents semantic content. Similar concepts produce similar vectors; different concepts produce distant vectors. The phrases "breach of fiduciary duty" and "violation of trust obligations" have embeddings close together in vector space, even though they share no words. This enables *meaning matching* rather than vocabulary matching. Embedding models vary in quality and domain fit; legal-specific models may outperform general-purpose ones for professional content. The infrastructure cost is higher: you need embedding models, vector

storage, and similarity search capabilities.

Structured queries retrieve from databases and APIs. An agent checking a company's SEC filings queries EDGAR; an agent researching a case retrieves from a court's electronic filing system. These are not "search" in the traditional sense—they are direct data access—but they serve the same function in RAG: finding relevant information to inject into context.

Hybrid approaches combine methods. A common pattern uses BM25 for initial retrieval (fast, cheap, catches exact matches) followed by semantic reranking (slower, more expensive, captures meaning). Many production systems blend keyword and semantic scores, getting the best of both approaches.

Choosing Retrieval Methods

No single retrieval method dominates. The right choice depends on your content, queries, and constraints:

- **Exact terms matter**: Keyword or BM25. Case citations, statutory references, and party names must match precisely.
- **Conceptual similarity matters**: Embeddings. Finding documents about "materiality" when the query says "significance" requires semantic matching.
- **Structured data**: Database queries or APIs. Retrieving a specific filing from EDGAR or a price from Bloomberg.
- **Production constraints**: BM25 is cheaper and faster; embeddings require more infrastructure but capture meaning better.

Most professional systems use hybrid approaches, combining methods to balance precision, recall, and cost.

Vector Stores for Embedding-Based Search. When using embeddings, storing and searching millions of vectors efficiently requires specialized infrastructure.

Vector Stores

A **vector store** (or vector database) is a system optimized for storing embeddings and performing similarity search. Given a query embedding, a vector store returns the most similar document embeddings from its collection. Vector stores use specialized indexing structures (like HNSW or IVF) that make approximate nearest-neighbor search practical at scale.

Vector stores are *infrastructure*—the retrieval mechanism that powers semantic search. They are not the same as the retrieval pattern itself.

Retrieval-Augmented Generation (RAG). With these retrieval methods in hand, we can now define RAG precisely.

Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) is a *prompt pattern*—not a software system or mathematical technique (Lewis et al. 2020). RAG augments a model's context with retrieved information before generation. The pattern has three steps:

1. **Retrieve**: Given a query, find relevant content using *any* retrieval method—keyword search, BM25, embeddings, database queries, API calls, or combinations thereof.
2. **Augment**: Insert the retrieved content into the model's prompt as additional context.
3. **Generate**: The model produces a response informed by both its training and the retrieved content.

RAG works because of in-context learning. The retrieved content appears in the prompt, and the model adapts its response to incorporate that specific information. The retrieval step is completely method-agnostic: what matters is getting relevant content into context, not how you find it.

The distinction matters. When evaluating a system, you should ask: What retrieval infrastructure does it use? (Keyword index, vector store, database, external API, hybrid?) What RAG pattern does it implement? (Single-stage, multi-stage, with reranking?) These are separate architectural decisions with different tradeoffs.

RAG Is Retrieval-Agnostic

A common misconception equates RAG with embeddings and vector stores. In practice, RAG implementations vary widely:

- **Keyword search**: Westlaw, Lexis, or internal full-text search—no embeddings required.
- **BM25**: Fast probabilistic retrieval powering many production systems.
- **Semantic search**: Embeddings and vector stores for meaning-based matching.
- **Structured queries**: SQL databases, EDGAR filings, court record APIs.
- **Hybrid**: Combining methods to balance precision, recall, and cost.

The "embedding plus vector store" pattern dominates tutorials because it is general-purpose and handles semantic similarity well. But many production systems—especially those leveraging existing search infrastructure—use simpler approaches that work well for their specific needs.

4.3 Model Context Protocol (MCP)

One of the persistent challenges in building agentic systems is integration. Every database, every document management system, every market data feed has its own way of accepting queries and returning results. Historically, connecting an agent to a new information source meant writing custom code for that specific system. Multiply that by the number of agents and the number of tools

an organization uses, and the integration burden grows quickly.

The Model Context Protocol (MCP) addresses this problem by standardizing how agents access tools (Anthropic 2024; "Model Context Protocol Specification" 2025). Instead of learning a different integration pattern for each system, agents learn the protocol once and can then access any compatible tool. For organizations, this means a single point of audit and control rather than dozens of bespoke connections to monitor.

The architecture has three components. The **MCP Host** manages the agent and controls what it can access, functioning like a firm's IT department deciding which systems a new employee can use. The **MCP Client** is the agent-side component that discovers available tools and makes requests. The **MCP Server** is what each information source runs to expose its capabilities through the standard interface. A document management system, an internal knowledge base, or a proprietary analytics platform can each run as an MCP server, and any MCP-compatible agent can then access them without custom integration work.

For perception specifically, MCP defines **Resources** as read-only data access endpoints. These might include document repositories, market data feeds, regulatory filing databases, or internal knowledge bases. The read-only designation matters: an agent with resource access can retrieve documents but cannot modify or file them. This separation enables fine-grained access control that mirrors how organizations already think about permissions.

The Integration Landscape Is Changing

Historically, many information providers in law and finance offered no programmatic access at all. Legal research platforms required human users working through web interfaces; market data terminals assumed a person at the keyboard. This created a barrier for agentic systems, which need machine-readable interfaces to function.

That landscape is shifting. Competitive pressure and customer demand are pushing providers to open up. Document management systems, e-discovery platforms, and financial data providers increasingly offer APIs that agents can use. Some legal research providers are beginning to follow. As this trend continues, the range of information sources available to agentic systems will expand significantly.

Standards like MCP accelerate this shift by reducing the cost of integration. Without MCP, connecting 10 agents to 10 tools requires 100 custom integrations. With MCP, the same setup requires only 20 implementations, since each agent and each tool learns the protocol once. Recent benchmarks found over ten thousand MCP servers already in the ecosystem (Mo et al. 2025), and that number will likely grow substantially in the years ahead.

4.4 Memory as Perception into Institutional Knowledge

Memory systems (Section 6) serve as perception tools for institutional knowledge. The retrieval concepts introduced above—embeddings, vector stores, and RAG—enable agents to perceive accumulated expertise that would otherwise be inaccessible.

When an agent queries a precedent database using the RAG pattern (Section 4.2), it perceives institutional knowledge through in-context learning. The retrieved content appears in the agent's prompt, allowing it to reason about specific precedents, prior analyses, and established approaches. A search for "breach of fiduciary duty" retrieves documents about "violation of trust obligations" because their embeddings are similar—not because they share keywords.

This mechanism enables perception into knowledge bases far too large to fit in any model's context window. A law firm's precedent database might contain decades of work product; a financial institution's research archive might span thousands of analyst reports. No agent can hold all of this in active context. RAG allows selective retrieval: the agent perceives only the most relevant fragments, guided by semantic similarity to the current task.

Institutional memory provides access to prior work product. When drafting a new registration statement, an agent can perceive prior S-1 filings (IPO registrations), SEC comment histories, and successful disclosure language. This access allows current work to build on verified precedents rather than starting from first principles.

Memory-as-perception distinguishes experienced agents from novices. A junior associate reasons from what they learned in law school; a senior associate draws on pattern recognition from hundreds of matters. Memory provides agents with this accumulated experience—but only if the retrieval infrastructure connects them to the right knowledge at the right time. Section 6 develops these requirements in detail, including how memory systems must enforce the authority, temporal validity, and isolation constraints introduced above.

4.5 Domain-Specific Perception Requirements

Perception for regulated professional services requires specialized enhancements. General-purpose search is insufficient; professional agents require authority tracking, jurisdictional awareness, and confidentiality boundaries.

Authority and Verification. Information varies in authority. Perception systems must track provenance to ensure reliability. Authority weighting ranks primary sources (statutes, binding precedent) above secondary sources (law reviews, news). When searching for "insider trading liability," a Supreme Court opinion outranks a commentary article. Source verification confirms that retrieved information originates from the claimed source. Perception tools must return verifiable citations, not just text. Currency validation ensures the authority remains valid. Integrated citators (like Shepard's or KeyCite) verify that retrieved cases have not been overruled.

Jurisdiction and Temporal Scope. Legal and regulatory information is bounded by jurisdiction. California precedent does not bind Texas courts; SEC rules differ from CFTC rules. Perception tools must filter results by relevant jurisdiction. Temporal validity is equally critical. Laws change, and financial data expires. Perception systems must track effective dates. In finance, validity varies by context: milliseconds for trading prices, quarters for compliance reporting. Identifier resolution manages the proliferation of formats. "123 F.3d 456" and "123 F3d 456" refer to the same case. Financial identifiers include tickers, CUSIPs, and LEIs (Legal Entity Identifiers). Perception must normalize these to ensure consistent retrieval.

Matter and Client Isolation.
Critically, perception must respect confidentiality boundaries. Whether a human or AI, an agent working on Matter A cannot perceive documents from adverse Matter B. This enforcement of **ethical walls** arguably must occur at the perception layer. In financial contexts, an agent advising Client X cannot perceive material non-public information (MNPI) from Client Y's engagement. Every perception event must be logged, capturing the agent, the query, and the matter context. This audit trail enables compliance review and breach detection. See Section 6 for detailed treatment of isolation requirements; *Governing Agents* addresses the professional responsibility obligations—including attorney confidentiality duties and fiduciary obligations—that mandate these controls.

4.6 Tool Design Principles

Robust perception tools follow design principles that enable reliable operation in professional environments.

Single Responsibility. Each tool should perform one function well. Poorly designed tools bundle multiple functions—searching, formatting, and validation—into opaque interfaces. Untyped return values obscure what callers can expect.

Poor Design: Bundled Functions, Untyped Returns

```
def legal_research(query: str) -> dict:
  """Returns... something. Good luck."""
  ...
```

When such a tool fails, diagnosing the error is difficult. A better approach separates tools by function with typed returns. This allows the agent to compose them and isolates failures.

Better Design: Single Responsibility, Typed Returns

```
def search_cases(query: str, jurisdiction: str) -> list[Citation]:
  """Returns matching citations from case law database."""

def retrieve_case(citation: Citation) -> CaseText:
  """Fetches full text for a specific citation."""

def shepardize(citation: Citation) -> CitatorResult:
  """Checks validity: good law, distinguished, overruled."""

def format_citation(case: CaseText, style: str) -> str:
  """Converts to Bluebook, ALWD, or other format."""
```

Graceful Failure. Production systems inevitably fail. Tools should return informative errors rather than generic exceptions. A poor approach raises exceptions that provide no context.

Poor: Opaque Exception

```
def retrieve_case(citation: str) -> dict:
  result = db.query(citation)
  return result["text"]  # raises KeyError if not found
```

A better approach uses typed result objects that make success and failure explicit.

Better: Typed Result with Structured Errors

```
class CaseNotFoundError(BaseModel):
  citation: str
  reason: str
  suggestions: list[str]

def retrieve_case(citation: Citation) -> CaseText | CaseNotFoundError:
  """Returns case text or structured error with recovery options."""
  if not (result := db.query(citation)):
    return CaseNotFoundError(
      citation=str(citation),
      reason="Case may not be in database",
      suggestions=["Check citation format", "Try alternative reporter"]
    )
  return CaseText(...)
```

In professional practice, graceful failure prevents malpractice. When an agent cannot find authority, it must report that explicitly rather than proceeding silently.

Least Privilege and Rate Limiting. Perception tools should request minimum necessary permissions. A legal research tool requires read access to case databases, not write access to the document management system. If a compromised agent gains perception credentials, damage is limited to information disclosure rather than destruction. Rate limiting addresses a common failure mode: infinite search loops. Tools should track invocation frequency and refuse requests beyond reasonable thresholds. If an agent searches five times without results, the tool should force a stop and escalation (Section 9).

4.7 Evaluating Perception Capabilities

When evaluating agentic systems, you should assess perception against criteria that matter for professional practice.

Coverage determines which sources the agent can access. A litigation agent that queries Westlaw but not state-specific databases has incomplete coverage. You must map available perception tools against information needs to identify gaps.

Retrieval quality measures whether the agent finds relevant information. Test with known-good queries where the correct result is established. Measure both **precision** (the fraction of retrieved documents that are actually relevant) and **recall** (the fraction of all relevant documents that the system successfully retrieves). These metrics will be familiar to legal professionals from technology-assisted review (TAR) in e-discovery, where courts have recognized precision and recall as the standard measures of retrieval effectiveness (Grossman and Cormack 2011). The same framework applies to agent perception: high precision means the agent does not waste time on irrelevant results; high recall means the agent does not miss important authorities. The tradeoff between them—casting a wider net improves recall but may reduce precision—is a design decision that should be calibrated to the task's risk profile.

Verification confirms that the system distinguishes authoritative from secondary sources. You must ensure that retrieved information is traceable to its source and that citations are independently verifiable.

Access controls ensure that permissions are appropriate. The agent must access only what it should, and confidentiality boundaries must hold across matter and client lines.

Failure handling reveals system behavior when perception fails. Does it retry, try alternatives, or escalate? It must not crash or proceed with incomplete information.

Audit capability confirms that every perception event is logged. You must be able to reconstruct what information the agent accessed during a task for compliance review.

4.8 From Perception to Action

Perception enables agents to gather information, but professional value ultimately requires effecting change: filing documents, sending communications, executing trades. This distinction between observing the world and changing it is fundamental to agent architecture, and the protocols we use reflect it. MCP, introduced earlier in this section, explicitly separates **Resources** (read-only data access) from **Tools** (operations that modify state). A single MCP server might expose both capabilities—a document management system could offer read access to files alongside the ability to create, modify, or delete them—but the protocol distinguishes these so that access control and governance can treat them differently.

The distinction matters because the consequences of failure differ fundamentally. When perception tools fail—a search returns wrong results, a document fails to load—the external world remains unchanged. The agent can retry, try alternatives, or escalate to human review without having caused any harm beyond wasted time. Action tools carry different stakes entirely. They file documents that become part of court records, send emails that reach recipient inboxes, execute trades that transfer ownership at market prices. Once executed, many actions cannot be undone, or can only be undone at significant cost. The agent's mistakes become facts in the world.

Section 5 examines action capabilities in detail, beginning with the conceptual foundations that distinguish actions from observations and developing the governance frameworks that these differences require.

5 How Does an Agent Make Things Happen?

A junior associate's role extends beyond research to producing work product. They draft memos, send emails, file documents, and schedule meetings. A trader's role extends beyond analysis to execution. They enter orders, route trades, and confirm allocations. Value derives from action, not just observation.

Agentic systems face the same imperative. An agent that only reads—searching databases, retrieving documents, analyzing information—produces no deliverable. To complete tasks, agents must *act*: generate documents, send communications, update systems, or execute transactions. Action implements the "A" in the GPA+IAT framework introduced in Part I (Bommarito et al. 2025).

Action Tools

Action tools enable agents to change the state of external systems. Unlike perception tools (Section 4), which are read-only, action tools *write*: they file documents, send messages, execute trades, and update databases. Once executed, some actions cannot be undone.

The distinction between perception and action is fundamental to governance. Perception risks involve internal errors (accessing wrong information). Action risks involve external consequences (harming clients, violating regulations, creating liability).

5.1 Conceptual Foundations of Action

Before examining specific action tools, we establish the conceptual foundations that distinguish actions from observations and that shape how we design, govern, and reason about them.

Actions as State Transitions. From a logical perspective, actions are state transitions: operations that move the world from one configuration to another. Before the filing, the motion was not on the court docket; after the filing, it is. Before the trade, the portfolio held one set of positions; after the trade, it holds another. This framing—common in computer science, philosophy, and AI planning research—highlights that actions have preconditions that must hold before they can execute and postconditions that describe the world after they complete.

This state-transition view raises a crucial question that philosophers and AI researchers call the **frame problem**: when an action occurs, what exactly changes, and what stays the same (McCarthy and Hayes 1969)? Filing a document changes the court record, but it should not change the client's contact information, the matter's billing status, or the contents of unrelated files. The frame problem is surprisingly difficult to solve in general, but well-designed action tools address it by making their effects explicit—specifying what they modify, what they preserve, and what conditions must hold for the action to succeed.

Performative Actions. Some actions are *performative* in the sense developed by philosophers of language (Austin 1962): they do not merely describe or report on the world but constitute changes to it. When a judge says "I sentence you to five years," the utterance does not describe a sentencing—it *is* the sentencing. When parties sign a contract, the signatures do not report that an agreement exists—they bring the agreement into existence.

Many professional actions share this performative character. Filing a motion is not a report about legal status; it is an act that changes legal status. Executing a trade is not an observation about ownership; it is a transfer of ownership. Sending a legal opinion creates reliance that may give rise to liability. This performative quality explains why action governance must be stricter than perception governance: an agent that retrieves the wrong document has made a research error that can be corrected, but an agent that files the wrong document has changed legal reality in ways that may be

difficult or impossible to undo.

Idempotency and Safe Retry. Computer science formalizes an important property of operations called **idempotency**: an operation is idempotent if performing it multiple times produces the same result as performing it once. Reading a file is idempotent—reading ten times leaves the file unchanged. Retrieving a case from Westlaw is idempotent. But sending an email is not idempotent: sending the same message ten times produces ten emails in the recipient's inbox. Executing a trade is not idempotent: executing the same order ten times produces ten separate transactions.

Idempotency matters enormously for error recovery and system reliability. When an agent's connection drops mid-operation, or when a timeout occurs before confirmation arrives, the agent faces a difficult question: did the action complete? For idempotent operations, the answer does not matter—the agent can safely retry. For non-idempotent operations, retry might cause duplicate actions with serious consequences: double payments, duplicate filings, repeated client communications. Robust action tools must either be designed for idempotency from the start—typically by using unique transaction identifiers that allow the system to detect and reject duplicates—or must provide explicit confirmation mechanisms that let the agent verify whether the action completed before deciding whether to retry.

Implications for Tool Design. These conceptual foundations have practical implications for how we design and evaluate action tools. Tools should document their preconditions (what must be true before the action can execute), their postconditions (what will be true after successful execution), and their effects on system state (what changes and what remains unchanged). Tools should indicate whether they are idempotent and, if not, what mechanisms exist to prevent duplicate execution. Tools should distinguish between actions that are truly performative—creating new legal or financial realities—and those that merely update internal records. These distinctions inform the governance frameworks developed throughout this section.

5.2 Action Tool Categories

Action tools vary in consequence. The critical dimension is **reversibility**: the cost and feasibility of undoing an action. Research on rollback-augmented systems demonstrates that selective state rollback reduces catastrophic failures in safety-sensitive environments (Grinsztajn et al. 2021). We categorize action tools along a spectrum from easily undone to permanent.

Communication tools send information to others. Internal communications (emails to colleagues, Slack messages) are *partially reversible.* You can follow up with corrections, though you cannot unsend. External communications (emails to clients, letters to counsel) carry higher stakes because recipients are outside your control. Retractions are possible but damage professional reputation. Automated alerts (compliance notifications, reminders) are generally low-risk if templated. Governance typically relies on post-hoc review for internal actions and pre-approval for external ones.

Document management tools create and organize work product. These are *largely reversible*. Drafting memos, generating reports, and filing documents in internal systems occur within the firm's control. Revisions are possible until distribution. Template application (populating standard forms) is low-risk if the templates are pre-validated. The primary control is review before external release.

Filing and submission tools send documents to external authorities. These are *largely irreversible*. Court filings via CM/ECF (Case Management/Electronic Case Files) become public record upon submission. Amendments are possible but the original remains visible. Regulatory submissions via EDGAR (Electronic Data Gathering, Analysis, and Retrieval) or FINRA systems trigger legal obligations. Errors may compel public corrections or enforcement actions. Contract execution creates binding legal obligations that are difficult to unwind. These actions require mandatory pre-approval.

Transaction execution tools transfer value or change ownership. These are *effectively irreversible* or costly to reverse. Trade execution involves entering orders and confirming allocations. Reversal requires offsetting trades at market prices, realizing any loss. Payment processing (wire transfers) moves funds immediately; recovery relies on recipient cooperation. System updates (modifying production databases) can disrupt operations. These actions demand the strictest controls: multi-factor approval, segregation of duties, and real-time monitoring.

5.3 The Reversibility Framework

Reversibility dictates oversight structure. Consider how you delegate to a junior associate: fully reversible work (research, drafting) proceeds independently with post-hoc review; partially reversible work (internal emails) gets checkpoint review; largely irreversible work (client communications, filings) requires pre-approval; and irreversible work (trades, wires) demands multi-party approval. Table 3 summarizes this mapping. Agent governance must enforce controls corresponding to each action's reversibility classification.

Table 3: Reversibility determines oversight

Reversibility	Examples	Oversight	Recovery
Fully	Research; drafts	Post-hoc	Delete/revise
Partially	Internal emails; alerts	Checkpoint	Correction
Largely irreversible	Filings; client emails	Pre-approval	Amend/retract
Irreversible	Trades; wires	Multi-party	Offset (costly)

This mapping from reversibility to oversight is not abstract—it directly determines which approval workflows apply. The following subsections operationalize these oversight tiers.

5.4 MCP Tools and Prompts for Action

The Model Context Protocol (MCP) defines two capability types relevant to action governance.

MCP Tools are executable functions that change state. Unlike read-only Resources, Tools create documents, send communications, submit filings, and execute transactions. Tool manifests should include risk metadata: reversibility classification, approval requirements, and audit logging needs. This allows the MCP Host to enforce controls automatically.

MCP Prompts are reusable templates for common tasks. For action workflows, prompts encode Standard Operating Procedures (SOPs).

- **Legal:** Contract review checklists, filing preparation workflows.
- **Finance:** Trade compliance checks, client onboarding sequences.

Prompts standardize action sequences, reducing variation and error. They act as "guardrails" by ensuring the agent follows the approved procedure for high-stakes actions.

5.5 Action Security

Every action interface is a security boundary. Actions access external systems and create real-world consequences.

All action tools must implement core security controls:

- **Authentication:** Verify the agent's identity via service accounts with strong credentials.
- **Authorization:** Enforce role-based access control (RBAC) and least privilege.
- **Input Validation:** Reject malformed requests by validating all parameters against strict schemas.
- **Output Confirmation:** Require human approval before executing high-stakes actions.
- **Rate Limiting:** Cap action frequency to prevent runaway execution.
- **Audit Logging:** Record every action with context (agent, timestamp, parameters, matter/client).

Beyond core controls, specific threats require targeted mitigations (OWASP Foundation 2025; Liu et al. 2024):

Prompt Injection via Action Parameters. Adversaries may embed malicious instructions in data that the agent processes and passes to tools. Mitigation requires sanitizing all parameters and never passing raw user input directly to sensitive action interfaces.

Privilege Escalation via Tool Chaining. An agent might combine multiple low-privilege tools to achieve a high-privilege outcome. Mitigation involves analyzing tool combinations and requiring approval for sequences that cross security boundaries.

Action Replay. An attacker might capture a valid action request (e.g., "Pay $100") and replay it multiple times. Mitigation requires **nonces** (unique, one-time numbers) or timestamps to ensure each request is processed only once.

5.6 Approval Workflows

For non-reversible actions, the agent prepares and the human approves (Parasuraman et al. 2000). These approval patterns operate alongside escalation logic (Section 9), which handles scenarios where human judgment is required regardless of the agent's confidence. We define three patterns for this division of responsibility.

The **Single Approver** pattern suits routine actions with clear authority. The agent completes preparation and presents it to one designated human. *Example:* The agent prepares a draft court filing; the supervising attorney reviews and approves; the agent submits.

The **Multi-Party Approval** pattern applies to high-stakes actions with significant exposure. Multiple independent humans must sign off. *Example:* The agent prepares a wire transfer. Operations reviews the amount. Compliance reviews the purpose. A manager provides final approval. Only then does the agent execute.

The **Escalating Approval** pattern adjusts authority based on risk tiers. *Example:* Trades under $100k require desk manager approval. Trades between $100k and $1M require senior trader approval. Trades over $1M require CIO approval.

Effective approval requests must enable informed decision-making. The agent should present:

- **Action:** Clear description of what will happen.
- **Context:** Why is this needed?
- **Risk:** What are the potential negative outcomes?
- **Reversibility:** Can this be undone?
- **Evidence:** What data supports this decision?

The approver should be able to decide based on the request alone, without needing to investigate the raw data. Note that idempotency design (discussed above) is critical here: if an approved action might be retried due to network failures or timeouts, the underlying tools must guarantee that re-execution does not cause duplicates.

5.7 Rate Limiting and Circuit Breakers

Agents can fail in loops: repeatedly submitting the same request, sending duplicate messages, or retrying failed transactions.

Rate Limiting caps action frequency. The thresholds below are *illustrative*; calibrate them to your workflow, risk tolerance, and regulatory obligations.

- *Per-Action:* e.g., max 5 emails per minute.
- *Per-Matter:* e.g., max 20 actions per day without review.
- *Cost:* e.g., max $1,000 in transaction fees per session.

When limits are reached, the agent must pause and escalate.

Circuit Breakers automatically stop execution upon anomaly detection. These examples are also *illustrative* and should be tuned to baseline behavior and threat models.

- *Failure Count:* e.g., if an action fails three times, stop.
- *Spike Detection:* e.g., if action rate spikes 5× above baseline, pause (potential compromise).
- *Cumulative Limit:* If daily total exceeds safety thresholds, lock the system.

Circuit breakers transform runaway failures into controlled pauses, buying time for human intervention.

5.8 Evaluating Action Capabilities

When evaluating agentic systems, assess action capabilities against professional standards.

Inventory: Map all available action tools against workflow requirements. Verify that no "unnecessary" tools are enabled (Least Privilege).

Classification: Verify that each tool is correctly classified by reversibility. Ensure that "delete database" is not classified as "fully reversible."

Controls: Confirm that approval gates exist for all irreversible actions. Verify that approvers receive sufficient context. Check authentication and audit logging.

Safety: Test rate limits and circuit breakers. Simulate a "runaway agent" scenario to ensure the system locks down. Verify rollback procedures: if an action fails, can you recover?

5.9 From Action to Governance

Action tools are where agentic systems create real-world consequences. Governance here differs in kind from perception governance.

Perception risks (accessing wrong data) are internal. Action risks (sending wrong emails, executing wrong trades) are external and potentially irreversible. This section established the architectural controls: the Reversibility Framework, approval workflows, and circuit breakers.

Section 6 addresses memory: how agents maintain context across sessions and learn from experience. Later, Section 9 examines when agents should *not* act—recognizing situations that require human decision-making. The interplay between action capability and escalation judgment is central to safety. These capabilities—action controls, memory, and escalation judgment—are then integrated into the governance architecture developed in Section 11. As *Governing Agents* emphasizes, governance policy without governance-aware architecture is unenforceable; the controls established in this section provide the foundation that policy requires.

6 How Does an Agent Remember Things?

The previous two sections examined how agents gather information (perception) and effect change (action). However, these capabilities operate in the moment: the agent perceives the current state, reasons about it, and acts. Without memory, every interaction resets. The agent cannot recall prior research, successful strategies, or case history. Memory transforms an agent from a stateless tool into a system that learns and adapts.

Every experienced professional knows that institutional memory distinguishes efficient work from reinventing the wheel. When starting a new securities registration or revisiting an investment thesis, you do not begin from scratch. You pull prior filings, review comment history, check precedent databases, and update existing models with new data. The firm maintains templates and research files that incorporate years of accumulated knowledge. Memory in agentic systems serves this same function: context retention across sessions and learning from experience, building on prior work rather than starting over.

Agent Memory

Agent memory stores and retrieves information across timescales (Park et al. 2023; Wang et al. 2024a).

- **Working Memory:** The documents actively loaded in the agent's context (like papers on a desk).
- **Episodic Memory:** The history of actions and outcomes for a specific matter (like a case file).
- **Semantic Memory:** The general principles and institutional knowledge available for retrieval (like the firm's precedent archive).

6.1 Memory Types: From Desk to Archive

Law firms use layered filing systems, each suited to different timescales. The associate's desk holds active work; the matter file contains engagement history; the precedent database archives institutional knowledge. Each layer trades immediacy for capacity. Agent memory systems follow the same pattern.

Working memory utilizes the **context window**—the text currently loaded in the LLM's active attention. Just like desk space, context windows have strict limits. An associate can only have so many documents open at once; an agent can only hold so many **tokens** (units of text, roughly 0.75 words) in active context. As of late 2025, leading models handle roughly 200,000 tokens. When the task exceeds this limit, you need other storage systems. In finance, this parallels the active trading screen: live prices and positions are immediate but transient.

Episodic memory corresponds to the matter file. Every memo, correspondence, and research result related to an engagement goes here. The associate does not re-research answered questions; the file provides the history. In agentic systems, episodic memory captures the log of actions and outcomes (Park et al. 2023). The agent records: "I searched for Ninth Circuit venue cases, found three opinions, and drafted the analysis." When asked a follow-up, the agent retrieves that prior state. This mirrors the financial research file: you pull prior analysis and update it, rather than starting fresh.

Semantic memory is the firm's precedent database. Institutional knowledge accumulates over decades. When you need a force majeure clause, the database offers fifty examples. Agentic systems access semantic memory through the RAG pattern introduced in Section 4.2: retrieve relevant content from a large corpus, inject it into the prompt, and generate a response informed by that specific knowledge (Lewis et al. 2020).

The retrieval infrastructure powering semantic memory—embeddings and vector stores—was defined in Section 4.2. Here, the key insight is that semantic memory works because of *in-context learning*: the model adapts its behavior based on what appears in its prompt. Retrieved precedents become part of the agent's working context, allowing it to reason about specific examples even when those examples were created after the model's training.

6.2 Implementing the RAG Pattern

The RAG pattern (Section 4.2) requires careful implementation to work reliably in professional contexts. The core pattern—retrieve, augment, generate—is simple, but the retrieval step admits many implementations with different tradeoffs.

The Core Pattern. RAG has three essential steps:

1. **Retrieve**: Find relevant content from your knowledge base.
2. **Augment**: Insert that content into the model's prompt.
3. **Generate**: Produce a response informed by the retrieved content.

The retrieval step is where implementations diverge. As discussed in Section 4.2, retrieval can use keyword search, BM25, embeddings, database queries, API calls, or hybrid combinations. The choice depends on your content, your queries, and your infrastructure.

The Embedding-Based Pipeline. When using embedding-based semantic search—the approach that dominates RAG tutorials—the retrieval step expands into a multi-stage pipeline:

1. **Chunking:** Breaks documents into semantic units (sentences, paragraphs, sections, or sliding windows) while preserving metadata (source, date, jurisdiction). Chunk size affects retrieval: too small loses context; too large dilutes relevance.

2. **Embedding:** Converts each chunk into a vector using an embedding model. Different models have different strengths; legal-specific embeddings may outperform general-purpose ones for professional content.

3. **Similarity Search:** Finds chunks similar to the query by comparing vectors (using cosine similarity or similar metrics). Returns the top-k most similar chunks.

This pipeline works well for unstructured text where semantic matching matters—finding documents about "materiality" when the query says "significance." But it is not the only approach, and for some use cases it is not the best.

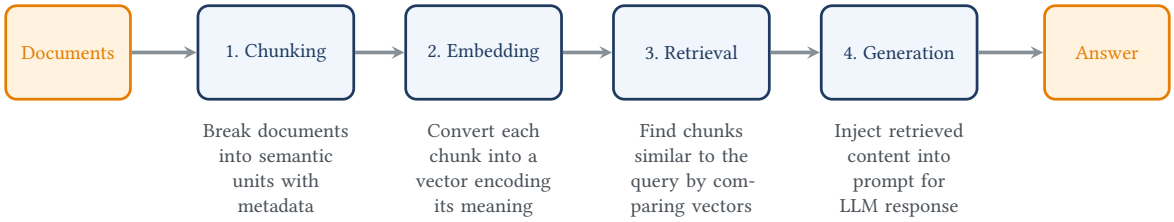


Figure 4: An embedding-based RAG pipeline—one common implementation pattern. Documents are chunked into semantic units, embedded as vectors, and retrieved by semantic similarity. Alternative implementations may skip chunking and embedding entirely, using keyword search, BM25, database queries, or hybrid approaches to retrieve relevant content.

Alternative Retrieval Approaches. Many production systems use simpler or hybrid approaches:

- **Keyword and BM25 search** requires no chunking or embedding infrastructure. Existing search systems—Westlaw, Lexis, internal document search—can power RAG directly. This works well when exact terms matter (case citations, statutory references) and vocabulary is consistent.
- **Structured queries** retrieve from databases or APIs. An agent checking SEC filings queries EDGAR; an agent researching court records queries PACER. No chunking or embedding required—the data source returns structured results.
- **Hybrid retrieval** combines keyword search (fast, catches exact matches) with semantic search (slower, catches conceptual matches) (Robertson and Zaragoza 2009). A common pattern uses BM25 for initial retrieval and embedding-based reranking to surface the most relevant results.
- **Knowledge graph retrieval** leverages structured networks of entities and relationships. Where embeddings capture semantic similarity, knowledge graphs encode explicit connections—corporate ownership hierarchies, regulatory supersession chains, case citation networks. When a query references a known entity, the agent queries the graph and traverses relationships through multi-hop reasoning, following paths that vector similarity alone cannot discover. This approach excels when domain structure matters: tracing precedential chains in case law, mapping corporate family trees for conflict checks, or navigating regulatory cross-references. We examine knowledge graph foundations and their applications to law and finance later in this book.

Enhancing Retrieval Quality. Regardless of retrieval method, several techniques improve precision and authority:

- **Query rewriting** transforms vague questions ("What's the rule?") into specific search queries based on conversation history and domain knowledge (Ma et al. 2023).
- **Reranking** scores initial results by authority or relevance, ensuring binding precedent ranks above secondary sources (Yu et al. 2024). A Supreme Court opinion should outrank a blog post, even if both match the query.
- **Filtered retrieval** constrains results by metadata—jurisdiction, date, document type—preventing agents from citing inapplicable authority.

Warning: The Hallucination Risk

Fabricated citations remain a critical failure mode even with RAG. Studies show that RAG-enabled tools can hallucinate 17–33% of the time (Dahl et al. 2024). The model may generate plausible-sounding citations that do not exist in the retrieved context—or anywhere.

You must implement verification: before any citation reaches the user, confirm that the source exists in the retrieved context. This is not optional for professional applications.

6.3 Domain-Specific Memory Considerations

Memory systems for regulated industries require enhancements that go well beyond generic implementations. Three dimensions matter most: authority, jurisdiction, and time.

Consider how a legal researcher thinks about sources. When investigating insider trading liability, a Supreme Court opinion carries far more weight than a law review article discussing the same topic. The human researcher instinctively applies this hierarchy; an agent's memory system must do the same. This means tagging documents with their position in the authority hierarchy—primary sources like statutes and binding precedent at the top, secondary sources like treatises and articles below—and ensuring that retrieval algorithms surface higher-authority documents more prominently. Financial systems face analogous challenges: an SEC no-action letter provides more reliable guidance than a client alert summarizing the same issue, and memory systems should reflect that difference.

Jurisdiction adds another layer of complexity. Legal information does not exist in a vacuum; it operates within territorial boundaries. California precedent does not bind Texas courts, and New York banking regulations have no force in London. A memory system that fails to account for these boundaries will produce results that are not merely unhelpful but actively misleading. When an agent researches Delaware corporate law, it must filter results to surface Delaware authorities as controlling while clearly distinguishing persuasive authority from other jurisdictions. This requires rich metadata tagging and retrieval logic that can enforce strict jurisdictional filtering when the task demands it.

Time presents perhaps the most challenging dimension. Law evolves through legislative amendments, regulatory updates, and judicial decisions that overturn or limit prior holdings. A case from 1985 may have been explicitly overruled, limited to its facts, or superseded by statute. Memory systems must integrate with citator services like Shepard's or KeyCite to validate that retrieved precedents remain good law. Financial data introduces even more varied temporal requirements: market prices become stale in milliseconds, earnings reports remain relevant for quarters, and industry analyses may hold value for years. Effective memory systems tag all data with effective dates and expiration windows, triggering refresh processes when content ages beyond its useful life.

Finally, professional domains create identifier resolution challenges that generic systems rarely encounter. Legal citations appear in multiple formats—"123 F.3d 456" and "123 F3d 456" refer to the same case, but a naive system might treat them as distinct. Companies accumulate multiple identifiers across different contexts: stock tickers, CUSIPs, ISINs, and Legal Entity Identifiers (LEIs) all point to the same entity but appear in different documents and databases. Without careful normalization, retrieval systems fail to connect related records, fragmenting information that belongs together.

6.4 Matter and Client Isolation

Perhaps no aspect of memory architecture matters more for professional services than enforcing strict **ethical walls** between matters and clients. The consequences of failure are severe: if an agent working on Matter A inadvertently accesses privileged information from Matter B—particularly when the matters involve adverse parties—the result may be privilege waiver, disqualification, and malpractice liability. Financial services face parallel risks when Material Non-Public Information (MNPI) leaks across the walls that separate investment banking from trading operations.

Implementing effective separation requires a layered approach that begins with architectural isolation. Each matter should occupy its own namespace within the memory system, creating a logical partition that separates its documents, notes, and work product from all other matters. Retrieval operations must be scoped to respect these boundaries: when an agent queries memory in the context of a particular matter, the system should only search within that matter's namespace, never reaching across into other partitions regardless of how relevant the results might appear.

Access controls provide the next layer of protection. Role-based permissions determine which agents and human users can access each namespace, mirroring the ethical wall policies that law firms and financial institutions already maintain for their human professionals. An associate staffed on a merger transaction should have access to that deal's namespace but not to the namespace for litigation against the same company being handled by a different team. When delegation introduces multiple agents accessing the same matter namespace (Section 10), isolation becomes even more critical: coordination among agents must not inadvertently leak information across matter boundaries.

Audit trails complete the picture by creating a verifiable record of every interaction with the memory system. Each read and write operation should be logged with a timestamp, the identity of the

requesting agent or user, and the matter identifier. These logs serve multiple purposes: they enable compliance review, support investigations if a breach is suspected, and provide documentation that the organization maintained appropriate controls.

Retention and deletion policies add a temporal dimension to isolation. Organizational policy often requires that matter files be retained for specified periods and then destroyed; legal and regulatory requirements may impose additional constraints depending on the engagement type, jurisdiction, and applicable rules. When a matter closes or a client relationship ends, the associated memory namespace should be archived or deleted according to a documented retention schedule. Critically, deletion must be verifiable—the organization needs confidence that purged data is truly gone, not merely hidden from normal retrieval. These isolation and audit requirements exemplify the architectural patterns for privilege enforcement and logging detailed in Section 11.1.

6.5 Evaluating Memory Systems

Memory evaluation requires four assessments: *retrieval quality* (precision and recall against expert work product—does it find the authorities a skilled attorney would cite?), *isolation integrity* (adversarial testing to verify matter/client boundaries hold), *temporal validity* (tracking whether retrieved precedent remains good law and how quickly updates propagate), and *scale performance* (latency as corpus grows to tens of thousands of documents). See the consolidated Evaluation Checklist in **??** for memory-specific assessment criteria including isolation integrity testing, retention policy verification, and temporal validity checks.

6.6 Adaptation: Learning and Its Consequences

Memory enables **adaptation**—the "A" in the GPA+IAT framework from *What is an Agent?*. An agent that adapts changes its behavior based on experience, ideally improving over time. This capability transforms agents from static tools into systems that learn. But adaptation also introduces risks that static systems avoid.

Adaptation

Adaptation is behavioral change based on experience. In agentic systems, adaptation occurs through three mechanisms:

- **In-context adaptation:** The agent changes behavior within a session based on instructions, examples, or retrieved content. This is in-context learning in action.
- **Cross-session adaptation:** The agent changes behavior across sessions by persisting information in memory and retrieving it later.
- **Model-level adaptation:** The model's weights are updated through fine-tuning or reinforcement learning. This is less common in deployed systems due to cost and complexity.

In-context adaptation is immediate and powerful. You provide the agent with examples of preferred

output format, and it adapts its responses accordingly. You correct an error, and subsequent responses reflect that correction. This is why retrieval matters: by controlling what appears in the agent's context, you control how it adapts.

Cross-session adaptation requires persistent memory. The agent stores information—user preferences, successful strategies, matter-specific context—and retrieves it in future sessions. This creates continuity: the agent "remembers" that this client prefers concise memos, that this portfolio manager wants risk metrics in basis points, that this matter involves a specific contractual provision.

Opportunities from Adaptation. Adaptation enables capabilities that static systems cannot match:

Personalization. The agent learns user preferences—communication style, level of detail, preferred formats—and tailors responses accordingly. A partner who always asks follow-up questions about procedural history gets proactive procedural context. An analyst who focuses on downside scenarios gets risk-weighted analysis by default.

Performance improvement. The agent learns from feedback. Corrections persist: "Actually, this jurisdiction uses different venue rules" becomes part of the agent's knowledge for future queries. Successful strategies are reinforced: the research approach that found relevant authority is remembered and reapplied.

Domain specialization. Through accumulated episodic and semantic memory, agents develop expertise in specific practice areas or market segments. An agent that has worked on dozens of M&A transactions "knows" the typical deal structure, common negotiation points, and relevant precedents—not through training, but through memory.

Continuity across handoffs. When a professional leaves or a matter transfers, institutional knowledge often walks out the door. Memory-enabled agents maintain continuity. The research history, strategic decisions, and accumulated context persist in retrievable form.

Risks from Adaptation. The same mechanisms that enable adaptation create risks that require governance:

Security: Memory poisoning. If an adversary can write to an agent's memory, they can influence future behavior (Clop and Teglia 2024). A document designed to be retrieved—containing misleading instructions or false information—can corrupt the agent's responses. This is prompt injection via the memory layer. The attack surface expands beyond the current conversation to include everything the agent might retrieve.

Security: Data leakage through memory. Memory that persists across sessions can leak information across contexts. If the agent stores sensitive information from Matter A and retrieves it during Matter B, confidentiality is breached. This risk intensifies with shared memory systems that serve multiple users or matters.

Governance: Behavioral drift. As agents adapt, their behavior changes in ways that may not be predictable or desirable. An agent that learns from user feedback might learn bad habits—shortcuts that work in routine cases but fail in edge cases. Accumulated memory can shift the agent's responses away from its original design.

Governance: Accountability gaps. When behavior depends on memory state, accountability becomes complex. Which version of the agent produced this output? What was in its memory at the time? If the agent's behavior changes based on accumulated experience, audit trails must capture not just inputs and outputs but memory state.

Detecting and responding to these risks—particularly drift and leakage—requires escalation triggers and monitoring mechanisms explored in Section 9, which addresses when anomalies warrant human review.

Cost: Storage and retrieval overhead. Memory consumes resources. Episodic memory for a complex litigation matter can grow to millions of tokens. Semantic memory for a precedent database can require terabytes of embeddings. Every retrieval operation has latency and cost. These costs compound as memory grows.

Latency: Retrieval time. RAG adds latency to every query. The agent must embed the query, search the vector store, retrieve results, and inject them into the prompt before generation begins. For real-time applications—trading, live client calls—this latency may be unacceptable.

The Adaptation Tradeoff

Adaptation is not optional. Any system that uses RAG, maintains conversation history, or persists user preferences is adapting. The question is not whether to allow adaptation, but how to govern it.

Key governance questions:

- **What can be written to memory?** Control the write path to prevent poisoning.
- **What can be retrieved?** Scope retrieval to prevent leakage.
- **How is memory validated?** Review accumulated memory for accuracy and appropriateness.
- **When is memory cleared?** Define retention policies and implement secure deletion.
- **How is drift detected?** Monitor behavior changes over time.

These governance questions are not abstract—they map directly to architectural controls developed in Section 11, which synthesizes logging, override mechanisms, and isolation patterns specifically designed to manage adaptation risks.

6.7 From Memory to Planning

Memory provides the context agents need to plan effectively. Without memory, agents cannot learn from failed strategies, cannot build on prior work, and cannot maintain the continuity that complex tasks require. The adaptation mechanisms discussed above (Section 6.6) transform memory from passive storage into active learning—but that learning must be channeled into productive planning.

Consider how an experienced associate approaches a new research task. They do not start from scratch; they recall similar matters, retrieve relevant precedents, and apply strategies that worked before. Memory-enabled agents can do the same: retrieve prior research, recall successful approaches, and avoid repeating failures.

Section 7 examines the next question: how does an agent break a big job into steps? Just as the case file enables strategic litigation planning, agent memory enables systematic task decomposition. The adaptation capabilities discussed here—personalization, domain specialization, performance improvement—all feed into more effective planning.

7 How Does an Agent Break a Big Job into Steps?

A litigation partner approaching a new matter understands that the path from initial consultation to trial and beyond is a long sequence of interdependent steps. To navigate it, the partner develops a strategy: discovery first (identifying needed facts), then dispositive motions when appropriate, followed by settlement discussions or trial preparation. Discovery breaks into phases: initial disclosures, document requests, interrogatories, and depositions. Tasks distribute across the team: a senior associate handles briefing, a junior associate reviews documents, and a paralegal manages scheduling. Throughout, the partner monitors progress against deadlines and adjusts strategy as new facts emerge.

This is **planning**: decomposing complex goals into action sequences. It mirrors the litigation roadmap or deal timeline that guides execution. Without planning, agents react to immediate observations without strategy. With planning, they work systematically toward objectives, adapt when circumstances change, and recognize completion.

Planning

Planning decomposes complex goals into sequences of actions (Russell and Norvig 2020). It encompasses:

- **Decomposition**: Breaking large tasks into manageable steps.
- **Sequencing**: Ordering steps logically based on dependencies.
- **Allocation**: Assigning steps to specific tools or sub-agents.
- **Monitoring**: Tracking progress toward the goal.
- **Adaptation**: Adjusting the plan when circumstances change.

Without planning, an agent resembles an associate running searches without a strategy—busy but not progressing toward a deliverable.

7.1 Planning Patterns

Three patterns dominate agent planning, each suited to different task types.

ReAct (Reasoning + Acting). The most fundamental pattern interleaves reasoning with action (Yao et al. 2022). Consider a partner asking for authority on an unenforceable forum selection clause. The associate reasons: "Key grounds are unconscionability and public policy. I will start with *Atlantic Marine*." They search, observe results, and reason again: "Unconscionability cases involve consumer contracts, not our commercial context. The public policy line is stronger." They search again, refining based on results.

Each cycle has three components:

- **Thought**: Explicit reasoning about what to do next.
- **Action**: A tool call to gather information or effect change.
- **Observation**: The tool output that informs the next thought.

Reasoning traces make decisions transparent and auditable. ReAct works well for exploratory tasks where you learn as you go—legal research, fact investigation, and market analysis.

Plan-Execute. This pattern separates planning from execution. For a document review task ("Review 50 contracts for choice-of-law provisions"), the associate creates a plan: list the contracts, open each one, extract the provision, and record findings. Then they execute systematically. The plan remains static because the task is well-defined.

Plan-Execute fits established workflows: due diligence checklists, compliance reviews, and document assembly. You create the plan upfront and execute methodically. Research variants like ReWOO (Xu et al. 2023) (separating reasoning from observation) and LLMCompiler (Kim et al. 2024) (optimizing execution graphs) enable parallel tool calling. However, the core pattern remains: plan first, then execute.

Hierarchical Planning. Law firms decompose matters into workstreams delegated through layers.

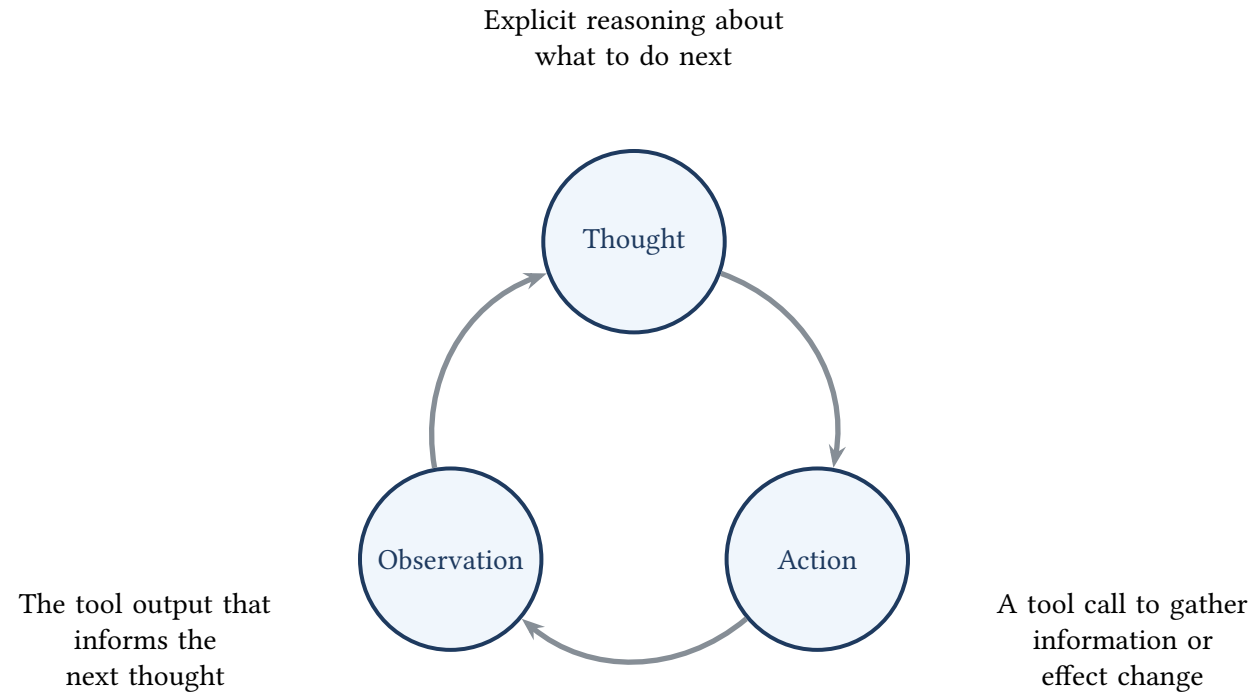


Figure 5: The ReAct cycle interleaves reasoning with action. The agent reasons about what to do (Thought), executes a tool call (Action), observes the result (Observation), and uses that observation to inform the next round of reasoning.

A parent agent receives a high-level goal, breaks it into sub-goals, and delegates to specialists. "Prepare for trial" becomes:

- Finalize witness list (delegated to Agent A).
- Prepare exhibits (delegated to Agent B).
- Draft witness questions (delegated to Agent C).

Each specialist may decompose further. This enables parallelization and specialization, mirroring how litigation teams work. Section 10 details these coordination patterns.

These patterns represent a shift from traditional workflow automation. Traditional engines used **static orchestration**: predefined graphs specifying exact steps and branches (BPM systems). The workflow was designed at build time; the engine merely executed it.

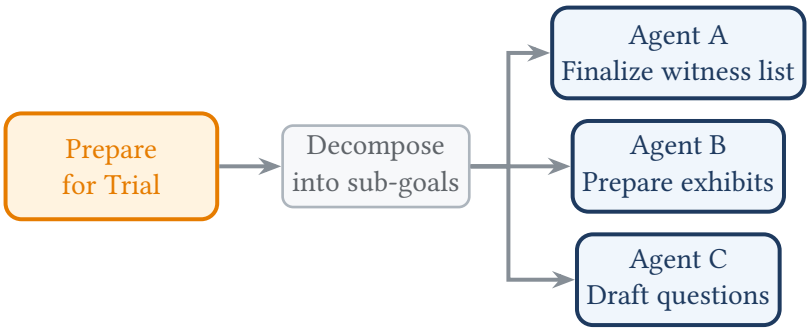


Figure 6: Hierarchical planning decomposes high-level goals into sub-goals delegated to specialist agents. Each specialist may decompose further, enabling parallelization and specialization.

Static vs. Dynamic Orchestration

Static orchestration executes predefined workflow graphs. The same input always produces the same execution path. It is predictable and auditable but inflexible.

Dynamic orchestration reasons about task decomposition at runtime. The LLM examines the goal, considers available resources, and constructs a plan on the fly. It is adaptive but less predictable.

LLM-based orchestration is inherently dynamic. "Prepare for trial" decomposes differently depending on case complexity. The LLM constructs a delegation structure based on the specific context. This adaptability is both the promise and the challenge.

Static workflows handle anticipated scenarios. Dynamic orchestration handles novel situations. Maintenance costs also differ: static workflows require explicit updates, while dynamic orchestration absorbs changes through prompt updates.

Auditability Challenge

Static workflows produce predictable execution traces. Dynamic orchestration may produce different decompositions for similar inputs, complicating audit. For regulated applications, you must log the *reasoning* behind orchestration decisions, not just the decisions themselves.

Production systems often combine both. High-volume, well-understood processes use static workflows. Complex, novel tasks use dynamic orchestration. The planning patterns described above—ReAct, Plan-Execute, Hierarchical—are forms of dynamic orchestration.

7.2 Choosing the Right Planning Pattern

Selecting the right pattern depends on task structure and required autonomy.

Table 4: Planning pattern selection guide

Task Type	Pattern	Autonomy	Example
Well-defined steps, known scope	Plan-Execute	Moderate	Credit review, compliance audit, due diligence checklist
Exploratory, learns as it goes	ReAct	Higher	Legal research, fact investigation, market analysis
Complex, parallel workstreams	Hierarchical	Distributed	M&A transaction, portfolio construction, multi-jurisdiction filing

Higher autonomy requires more sophisticated oversight.

Plan-Execute operates with moderate autonomy. The agent works within bounds defined by the plan. Oversight focuses on plan validation. ReAct involves higher autonomy because the agent decides what to search and when to stop. Oversight requires explicit termination mechanisms and confidence thresholds; these thresholds also feed into escalation decisions (Section 9) that determine when uncertain decisions should transfer to humans. Hierarchical patterns distribute autonomy. Oversight requires clear delegation contracts and escalation paths. You must match oversight rigor to autonomy level.

7.3 Understanding the Task Before Planning

Before an agent can construct a plan, it must develop a clear understanding of what it has been asked to accomplish. While Section 3 explores intent extraction in detail, the planning system depends on three specific outputs from that process: a classification of the task type, an understanding of the constraints that bound acceptable solutions, and criteria that define what success looks like.

Task classification shapes the choice of planning pattern. An exploratory question—"What are the key risks in this contract?"—calls for the flexible, iterative approach of ReAct. A well-defined multi-step procedure—"Review this document against our standard checklist"—fits the structured decomposition of Plan-Execute. Complex engagements with nested subtasks and dependencies require hierarchical planning with explicit coordination. Choosing the wrong pattern wastes resources or produces inadequate results.

Constraints establish the boundaries within which the plan must operate. These include explicit limits like deadlines and budgets, but also implicit bounds like the scope of the engagement and the level of detail expected. A request to "summarize the key terms" differs fundamentally from a request to "conduct comprehensive due diligence," even if both involve the same underlying document.

Success criteria complete the picture by defining how the agent recognizes that its work is done. Without clear criteria, agents struggle to terminate appropriately—they may stop too early, leaving

important questions unanswered, or continue indefinitely, consuming resources on diminishing returns. The clearer the goal, the more focused the plan; ambiguity at the input stage propagates through the entire execution.

7.4 Budget Architecture

Before examining resource budgets, note that memory constraints shape planning from the start. Working memory—the agent's context window—has strict limits, currently around 200,000 tokens for leading models (Section 6.1). A plan that assumes unlimited context will fail when the agent cannot hold all relevant information simultaneously. Similarly, if the agent must retrieve information from episodic or semantic memory, retrieval cost and latency become part of the plan's resource budget. Memory is not merely storage; it is a planning constraint.

Without resource constraints, agents can run indefinitely—the computational equivalent of asking a junior associate for two relevant cases and receiving fifty, along with a bill for the hours spent finding them. Budget architecture provides the planning mechanism that prevents this outcome, giving agents explicit limits that shape how they allocate effort across tasks.

Resource consumption in agent systems takes several forms, each requiring its own type of constraint. Token budgets limit consumption of LLM API calls, preventing expensive reasoning loops where the agent repeatedly processes the same information or explores unproductive tangents. Time budgets enforce deadlines by halting execution after a specified duration, ensuring that a task expected to take ten minutes does not silently expand to consume an hour. Tool call budgets cap external interactions—limiting an agent to twenty database searches, for instance, forces it to formulate queries carefully rather than issuing dozens of slightly varied requests. Cost budgets provide the most direct control, capping total spending in dollars regardless of how that spending is distributed across tokens, tools, and time.

These budgets operate hierarchically, cascading from broad constraints down to specific allocations. A session budget might constrain an entire client engagement; within that envelope, individual tasks receive their own allocations, which subdivide further into budgets for subtasks and individual operations. A legal research task might receive thirty minutes and fifty thousand tokens; the subtasks that compose it—identifying relevant statutes, finding controlling cases, synthesizing holdings—share this pool rather than each receiving unlimited resources.

Understanding how costs compound is essential for realistic budgeting. Ingesting a 200-page credit facility consumes roughly 80,000 tokens before any analysis begins. The analysis itself requires additional tokens for reasoning, and a comprehensive review of a complex document might reach one million tokens in aggregate. Monitoring must track cumulative consumption across the entire task, not just individual operations, because costs that seem modest in isolation can accumulate rapidly.

The economics of agent assistance vary significantly by task type. Retrieval-heavy work like docu-

ment review often shows clear return on investment: the agent processes material faster and more consistently than a human reviewer, and the cost savings are straightforward to calculate. Judgment-intensive tasks present a more complex picture, since extensive human review of the agent's output may be required, potentially reducing the net benefit. Transparency about AI assistance—as encouraged by recent ethics guidance (American Bar Association Standing Committee on Ethics and Professional Responsibility 2024)—enables clients to evaluate this value proposition for themselves.

Well-designed agents degrade gracefully as budgets tighten rather than failing abruptly. The key is tiered output that provides value at every resource level. With a minimal budget, the agent might return only the controlling statute. A moderate budget allows it to add key holdings from relevant cases. A full budget enables comprehensive analysis with supporting citations and counterarguments. Soft limits—triggered at perhaps eighty percent of the allocated budget—warn the agent to prioritize completion over thoroughness. Hard limits at one hundred percent terminate execution and return whatever results have been assembled. An agent that delivers useful partial results within budget is far more valuable than one that produces nothing when resources run short. When budgets are exhausted, the agent must terminate. The critical design principle is that termination under budget pressure should still produce value: the agent delivers what it has discovered rather than nothing. Section 8 addresses how termination conditions interact with budget limits, including how to define success for partial results.

7.5 Stopping Conditions

Planning must include stopping conditions. Success criteria, resource limits, and confidence thresholds all determine when an agent should stop. Section 8 addresses termination in depth; here we note that the planning phase is when these conditions must be defined. A plan without stopping rules is incomplete—the agent will either terminate prematurely or continue indefinitely, consuming resources without adding value.

7.6 From Planning to Termination

Planning decomposes work, but every plan must end. The next two questions address the boundaries of autonomous execution.

Section 8 (Termination) addresses how an agent knows it is done. This requires defining success criteria and budget limits. Section 9 (Escalation) addresses how an agent knows when to ask for help. This requires confidence thresholds and authority boundaries.

Without clear termination, agents run forever. Without escalation, they exceed authority. These boundaries define the safe operating envelope for autonomous systems.

8 How Does an Agent Know When It's Done?

Every professional learns to recognize completion. The research memo is done when you have found sufficient authority and synthesized it. The due diligence is done when you have reviewed all material documents. The trade is done when the order executes and settles. Knowing when work is complete distinguishes effective professionals from those who over-research or under-deliver.

Agents face the same challenge. Without explicit termination conditions, agents can run indefinitely: searching one more database, trying one more approach, refining one more time. We call this the "runaway associate" problem: you ask for two relevant cases, and the associate gives you fifty because they did not know when to stop.

Termination

Termination conditions define when an agent should stop executing. Three outcomes are possible:

- **Success:** The goal is achieved, and the agent delivers the result.
- **Failure:** The goal cannot be achieved; the agent reports why and stops.
- **Escalation:** The agent cannot determine success or failure, transferring the decision to human judgment.

Termination implements the "T" in the GPA+IAT framework introduced in *What is an Agent?*. Without it, agentic systems lack the property that distinguishes systems from runaway processes.

8.1 Termination Condition Categories

Five categories of termination conditions bound agent execution, each addressing a different aspect of when and why an agent should stop working autonomously.

Success conditions represent the ideal outcome: the agent terminates because it has achieved its goal. But recognizing success requires clear criteria. *Completeness* asks whether all required items have been addressed—have all fifty contracts in the review queue been analyzed? *Quality* asks whether the output meets the required standard—are conclusions supported by binding authority rather than secondary sources? *Convergence* recognizes when continued effort yields diminishing returns—if three consecutive searches produce no new relevant authority, the research space is likely saturated. Quality assessment often requires human validation, but completeness and convergence can frequently be evaluated programmatically.

Resource budgets provide hard limits that prevent runaway execution. Section 7.4 details budget architecture—token limits, time limits, tool call caps, and cost ceilings. The termination implication is straightforward: when any budget is exhausted, the agent must stop. Budget exhaustion is not

necessarily failure; partial results assembled before the limit may be valuable and should be preserved. This principle is formalized in Section 8.6 as tiered output design.

Confidence thresholds gate autonomous action on the agent's certainty about its conclusions. When confidence is high, the agent delivers its answer and terminates normally. When confidence drops below a specified threshold—perhaps eighty percent for routine matters, higher for consequential decisions—the agent should stop and escalate rather than proceeding with uncertain conclusions. This mirrors the behavior expected of a well-trained associate: "I've found relevant authority, but I'm not confident it controls here. Let me ask the partner before we rely on this." Calibrating these thresholds presents a genuine challenge, as research has shown that language models can be systematically overconfident in their outputs (Kadavath et al. 2022). Effective calibration requires testing against known outcomes and adjusting thresholds based on observed reliability.

Error conditions require agents to recognize when something has gone wrong and continued execution is unlikely to help. *Repeated failures*—a database that times out on three consecutive attempts, an API that returns malformed responses—indicate problems that retrying will not solve. *Inconsistent data*, such as revenue figures in a 10-K that conflict with the earnings release, suggests either an error in the source documents or a parsing problem that requires human investigation. *Constraint violations* demand immediate termination: if a planned trade would exceed position limits or a proposed filing would miss a regulatory deadline, the agent must stop before taking the problematic action. Perhaps most important is recognizing *impossibility*—when requirements genuinely conflict or the requested task cannot be completed as specified, the agent should report this finding rather than compromising on some requirements to satisfy others.

Escalation triggers require human judgment regardless of whether the agent has succeeded or failed at its immediate task. Novel situations without clear precedent, high-stakes decisions with significant consequences, and actions that would exceed the agent's authority boundaries must all trigger handoff to human oversight. Unlike termination with failure, escalation *pauses* the task and requests human input before continuing—the work may resume once the human provides guidance. Section 9 examines when escalation (rather than termination) is the appropriate response.

8.2 Defining Success Criteria

While Section 7.4 addressed resource limits—the *hard* stopping points that prevent runaway execution—success criteria address the complementary question: how does the agent recognize that its work is *complete*? Vague goals produce unclear termination. An instruction to "research the statute of limitations" leaves the agent uncertain about scope, depth, and format. Which statute of limitations—for what claims, in which jurisdiction? How many sources are enough? What form should the output take? Without answers to these questions, the agent cannot recognize when its work is complete.

Effective success criteria take several forms, often used in combination. **Completeness checklists** enumerate the specific deliverables required. A credit agreement review might specify that the agent

must identify all financial covenants, compare each to market terms from a reference database, and summarize material risks in a structured format. The agent terminates only when every item on the checklist has been addressed, providing a clear and verifiable completion signal.

Sufficiency thresholds define what "enough" means for tasks without exhaustive requirements. Legal research rarely requires finding every case ever decided on an issue; instead, sufficiency might mean identifying three on-point opinions from the controlling circuit, or finding both the majority rule and any significant minority positions. Once the threshold is reached, the agent stops rather than continuing to search every available database. This prevents over-research while ensuring adequate coverage.

Convergence criteria recognize when continued effort produces diminishing returns. If three consecutive searches using varied query strategies yield no new relevant results, the research space is likely saturated. The agent can terminate with confidence that additional searching would not materially improve the analysis. This approach works particularly well for exploratory tasks where the scope cannot be fully specified in advance.

Deliverable specifications define the expected output format, which itself signals completion. "Produce a two-page memorandum with an executive summary, statement of facts, analysis, and conclusion" tells the agent exactly what success looks like. When the document matches the specification, the task is done.

The most effective approach combines these criteria in instructions that read like guidance to a junior associate:

> *"Research the statute of limitations for breach of fiduciary duty claims in Delaware. If you find clear Court of Chancery authority, you are done. If the courts have split or the issue is unsettled, map the competing positions. If you find nothing on point after searching for two hours, stop and report that the issue may be novel. Deliver your findings in a one-page summary with citations."*

8.3 Recognizing Failure

Agents must recognize and report failure honestly, resisting any tendency to mask problems or present incomplete work as complete. This requires a cultural shift in how we think about agent outputs: *negative results are valuable information*, not embarrassing admissions. "I searched all available databases using multiple query strategies and found no authority on point" is a legitimate and useful finding—it suggests the issue may be novel, the search terms may need refinement, or the legal theory may lack support.

The difference between useful and useless failure reports lies in **diagnostic detail**. "Task failed" tells the human supervisor nothing actionable. A proper failure report explains what was attempted and why it did not succeed:

> *"I searched Westlaw and Lexis using the following queries: [list]. Westlaw returned twelve results, none addressing the specific issue of whether the duty extends to indirect subsidiaries. Lexis returned zero results. The absence of authority suggests either that the issue is novel, that practitioners use different terminology, or that the question is typically resolved through contract rather than litigation. I recommend manual review with an expanded search strategy."*

This report enables the supervisor to decide whether to try different approaches, consult additional resources, or conclude that the absence of authority is itself the answer.

Partial completion must be preserved and clearly reported. If an agent was reviewing ten articles in a contract and encountered a failure after completing four, it should not discard its work. Instead, it should report: "Analysis complete for Articles 1 through 4; results attached. Articles 5 through 10 remain unanalyzed due to [reason for failure]." This preserves the value already created and gives the supervisor a clear picture of what remains.

Root cause identification aids the human response by distinguishing between *transient* and *structural* problems. A database timeout is likely transient—waiting and retrying may succeed. A parsing error on a malformed document may require human intervention to obtain a clean copy. Fundamentally conflicting requirements are structural—no amount of retrying will resolve them. The agent's diagnosis of the failure mode directly informs what the supervisor should do next.

8.4 Guardrails and Loop Detection

Even well-designed termination conditions cannot prevent every failure mode. Agents can become trapped in unproductive loops—repeating the same actions, cycling through equivalent states, or making nominal progress that adds no real value (Zou et al. 2024; Ma et al. 2024). Production systems require explicit guardrails to detect and interrupt these patterns; Section 11.2 addresses how circuit breakers implement these guardrails at the governance layer.

Step limits provide the simplest and most reliable backstop. Regardless of other conditions, after N total steps the agent must stop and require human approval before continuing. This prevents unbounded execution even when other detection mechanisms fail. The appropriate limit depends on the task: a simple lookup might warrant only ten steps, while complex research might allow a hundred. The key is that *some* limit exists and is enforced.

Progress detection monitors whether recent actions have produced value. If the last five tool calls returned no new information—the same documents retrieved, the same search results, the same analysis repeated—the agent is likely stuck in a loop or has exhausted productive avenues. This pattern should trigger either a reflection step (if the agent has that capability) or escalation to human oversight. Progress detection requires defining what "new information" means for each task type, which can be as simple as tracking whether retrieved documents have been seen before.

Reflection steps give agents the opportunity to assess their own behavior. Periodically, or when triggered by apparent lack of progress, the agent pauses to ask itself: "Am I making progress toward the goal? Have my recent actions been productive? Should I try a different approach, or is it time to stop and report what I've found?" This metacognitive capability is not yet reliable in all models, but when it works, it can catch problems that simple pattern matching would miss.

External watchdogs monitor agent behavior from outside the agent's own reasoning process. A watchdog might detect that the same tool has been called repeatedly with identical or near-identical parameters—a clear sign of a loop—and intervene to halt execution. Watchdogs can also enforce patterns that would be difficult to specify within the agent's instructions, such as rate limits on expensive operations or detection of oscillating behavior where the agent alternates between two states without progressing. Without some form of loop detection, agents deployed in production will eventually get stuck, potentially consuming significant resources before anyone notices.

8.5 The Reliability Cliff

Benchmarking reveals a striking pattern: agent performance does not degrade gradually as tasks become more complex, but instead *falls off a cliff*. METR's 2025 research found near-perfect success on tasks completable in minutes, but under ten percent success on multi-hour tasks (METR 2025). The exact boundary shifts as models improve, but the underlying design challenge persists.

The Reliability Cliff

This cliff reflects **compounding errors** (Press et al. 2023): each step has some probability of failure, and these probabilities *multiply*. A chain of twenty steps at 95% per-step reliability yields only 36% end-to-end success. Two factors steepen the cliff: **planning fragility** (early errors propagate forward and invalidate subsequent work) and **integration brittleness** (API timeouts and rate limits accumulate over time).

Design for this reality: Decompose aggressively. Keep agent tasks short. Insert human checkpoints. Build for resumability. And test against your own workflows—benchmark results establish a baseline, but your specific tasks may differ from standardized test suites.

8.6 Graceful Degradation

When termination occurs before task completion—whether due to budget exhaustion, confidence drops, or error conditions—the agent should not simply stop and report failure. Instead, it should *degrade gracefully*, delivering whatever value it has accumulated and positioning the human supervisor to continue effectively.

The key to graceful degradation is **tiered output design**. Rather than treating tasks as all-or-nothing, effective agents structure their work to provide value at multiple levels of completion. Consider a

legal research task: with minimal resources, the agent might deliver only the controlling statute and its citation—a modest but genuinely useful result. With moderate resources, it adds the key holdings from relevant cases, providing context for how courts have interpreted the statute. With full resources, it delivers comprehensive analysis including counterarguments, circuit splits, and practical implications. Each tier represents a complete, usable work product rather than a fragment of an unfinished whole. This structure allows the user to assess whether the partial result suffices or whether additional investment is warranted.

Progress preservation ensures that early termination does not waste the work already completed. If an agent stops mid-way through a contract review, it should save its state in a form that allows either itself or a human to resume without starting over. The four articles already analyzed should not require re-analysis; the search queries already executed should not need re-running. This requires deliberate architectural choices—checkpointing intermediate state, maintaining audit trails of completed steps, and structuring tasks as resumable sequences rather than monolithic operations.

Clear status reporting transforms an incomplete result into an actionable handoff. Rather than a bare "Task incomplete," the agent should report its progress precisely: "Completed analysis of Articles 1 through 4 (60% of task). Remaining: Articles 5 through 8. Findings so far: two material deviations from market terms identified in covenant structure." Critically, the agent should also provide a *recommendation* for next steps: "Recommend allocating 30 additional minutes to complete the remaining articles, or proceed with partial findings if timeline requires." This enables informed human decision-making about whether to invest additional resources, proceed with partial information, or reassign the task entirely.

8.7 Evaluating Termination Capabilities

Termination evaluation requires six assessments: *success clarity* (are termination conditions explicit and predictable?), *budget enforcement* (are limits actually respected, not exceeded by 20%?), *loop detection* (does it escape unproductive patterns?), *failure reporting* (do error messages enable effective human follow-up?), *graceful degradation* (do partial results have standalone value?), and *escalation handoff* (does the human receive sufficient context to continue?). See the consolidated Evaluation Checklist in **??**.

8.8 From Termination to Escalation

Termination and escalation are closely related but distinct concepts. Termination defines *when* an agent stops; escalation defines *what happens next* when stopping requires human involvement. An agent that terminates successfully has completed its task and can deliver results. An agent that terminates due to failure has determined that the task cannot be completed and reports why. But an agent that *escalates* has reached a different conclusion: not "I'm done" or "This is impossible," but rather "I need help to proceed."

This third category is critical for safe deployment in professional contexts. An agent researching a legal question might find genuinely conflicting authority that requires judgment to reconcile. An agent reviewing a contract might encounter a provision outside its training distribution that it cannot confidently interpret. An agent executing a financial transaction might face a decision that exceeds its authorization limits. In each case, the correct response is neither to forge ahead (risking error) nor to report failure (abandoning recoverable work), but to pause and request human input. This is not failure—it is *safety*.

Section 9 examines escalation in depth: when should an agent stop autonomous operation and ask for help? What information should it provide to the human taking over? How should escalation thresholds be calibrated for different risk levels? Together, termination and escalation define the complete boundary of autonomous execution. Without robust termination, agents run indefinitely. Without appropriate escalation, they exceed their authority or make decisions they are not qualified to make. Both capabilities are essential for trustworthy deployment. Section 11 examines how escalation hooks must be architecturally implemented to ensure they are reliable and cannot be bypassed.

9 How Does an Agent Know When to Ask for Help?

Sometimes the best thing to know is when you do not know something. A wise junior associate knows when to consult their supervising partner. In turn, a deputy corporate counsel can often tell when an issue should be escalated to someone more senior within the organization. It is a balancing act: it is unwise to interrupt a partner or senior lawyer with every question, but it is also unwise to proceed confidently into territory beyond your expertise. In a sense, the most effective junior professionals recognize authority boundaries: "I can draft this motion, but I need partner review before filing." They recognize competence limits: "I have researched for two hours and cannot find clear authority—I should ask someone with more experience." They recognize high-stakes situations: "The client is asking about strategy, not just research—this needs partner involvement."

Agentic systems require the same judgment. An agent that never escalates will eventually exceed its competence, authority, or the bounds of safe autonomous operation. An agent that escalates everything provides no value: it becomes a complicated way to route work to humans. The challenge is drawing the line.

Escalation

Escalation transfers control from the agent to a human when autonomous execution should stop. Unlike termination, which ends the task (success or failure), escalation pauses the task and requests human input before continuing.

This reflects professionalism, not failure. Recognizing when you need help and asking for it is exactly what we expect from junior professionals. Agents should do the same. Escalation operationalizes the "T" (Termination) element of the GPA+IAT framework from *What is an Agent?*, representing the third outcome—alongside success and failure—when autonomous execution should stop.

9.1 When to Escalate

Three categories of triggers warrant escalation, each reflecting a different reason why autonomous action should pause (Mosqueira-Rey et al. 2023; Gomez et al. 2025). These complement the termination conditions in Section 8 and build on the planning patterns in Section 7 that decompose work into steps where escalation decisions can be made.

Mandatory triggers require human involvement regardless of the agent's confidence or the apparent simplicity of the decision. Some situations demand human judgment by their nature, not because the agent is uncertain. When resource limits approach exhaustion, the agent should escalate with a progress summary rather than stopping silently—the human needs to decide whether to allocate additional resources or accept partial results. High-stakes actions like court filings, client communications, and large trades require human approval before execution; these are *approval gates* where the agent prepares but does not act. Actions that would exceed the agent's authorized thresholds—a trade above a certain size, a commitment beyond a certain value—require human sign-off even if the agent is confident the action is correct. And irreversible actions warrant particular caution: once a motion is filed or a trade is executed, the consequences cannot be undone, making pre-execution review essential. This principle aligns with the reversibility framework in Section 5.3, which maps oversight intensity to action consequences.

Confidence-based triggers occur when the agent's uncertainty exceeds acceptable thresholds for autonomous action (Madras et al. 2018; Abbasi Yadkori et al. 2024). An agent researching a legal question might find genuinely conflicting circuit authority and be unable to determine which rule applies—"I found three circuits supporting one approach and two supporting another, and I cannot determine which controls here." Data sources might disagree in ways the agent cannot reconcile: the revenue figure in the 10-K differs from the earnings release, and the agent cannot determine which is correct or whether both are valid in different contexts. Novel situations—unprecedented fact patterns, unusual market conditions, first-impression legal questions—push beyond the agent's training distribution, where its confidence estimates become unreliable. And sometimes ambiguity persists despite the agent's attempts to clarify: the instructions can be read multiple ways, and the agent remains uncertain which interpretation the human intended.

A critical implementation challenge: LLM-reported confidence scores are notoriously unreliable—models often express high confidence in incorrect answers (Kadavath et al. 2022). Rather than trusting model "vibes," production systems should use *system-level confidence signals* derived from observable properties of the agent's work.

Table 5: Operational confidence signals for escalation decisions

Signal Type	What to Measure	Escalation Trigger
Retrieval quality	Source authority, recency, topical coverage	Fewer than N authoritative sources found
Source agreement	Cross-reference consistency across retrieved documents	Contradictory information from comparable sources
Validation success	Schema validation, citation verification, data reconciliation	Citator shows "overruled" or "questioned"; figures do not reconcile
Tool reliability	API response codes, timeout rates, structured error returns	Tool calls failed or returned malformed data
Self-consistency	Agreement across multiple independent samples	High variance in answers when sampling the same question
Boundary detection	Pattern matching against known edge cases or out-of-scope indicators	Query matches escalation rules (e.g., mentions privilege, MNPI)

These signals are architectural, not learned—they derive from the structure of the agent's perception and action systems rather than from the model's internal states. A legal research agent that finds only two cases on point, both from trial courts, has low retrieval quality regardless of how confidently the model phrases its summary. A financial agent whose data sources show a 15% discrepancy in reported revenue has detected source disagreement that warrants human review. These signals are measurable, auditable, and far more reliable than asking the model "how sure are you?"

Error and anomaly detection triggers escalation when the agent encounters problems it cannot resolve through retrying or alternative approaches. Repeated failures—a database that times out on three consecutive attempts, an API that returns errors despite varied request formats—indicate systemic problems that further attempts will not solve. Data anomalies, such as financial figures that do not reconcile or filing dates that seem implausible, warrant human investigation rather than silent acceptance or rejection. Constraint violations require immediate escalation: if executing a planned action would breach position limits, exceed authorization thresholds, or violate compliance rules, the agent must stop and confirm rather than proceed. And when a task proves genuinely impossible—requirements conflict, necessary data does not exist, or the goal cannot be achieved as specified—the agent should report this finding rather than struggling indefinitely.

9.2 How to Escalate

Effective escalation provides the human with everything needed to make a decision without starting from scratch. The difference between good and poor escalation is the difference between receiving a well-organized file from a colleague and inheriting an unexplained mess.

A complete handoff weaves together five elements: a **situation summary** that orients the human quickly; a **progress report** explaining what has been done and what remains; a **trigger explanation** clarifying why the agent is escalating *now*; the **gathered information** presenting relevant findings even if partial; and a **recommendation** proposing a path forward with supporting reasoning.

Consider a legal research agent investigating the statute of limitations for a Section 10(b) securities fraud claim. After searching Westlaw and Lexis, it finds clear authority on the two-year discovery period but conflicting circuit authority on the trigger event—the Ninth and Second Circuits apply different tests for inquiry notice. The agent cannot determine which test applies to the client's facts. Rather than guessing, it escalates with a summary of the key cases, explains the circuit split, and recommends seeking partner guidance because the question is fact-intensive and requires judgment beyond the agent's competence.

A financial agent faces a different escalation pattern. After generating a trade list to reduce tech exposure from 35% to 25%, the agent completes its compliance check and is ready to execute—but the $500K trade size exceeds the single-approver threshold. The agent escalates with the tax implications ($45K in short-term gains, $12K in losses, $8K net liability), presents options (approve the full list, prioritize tax-loss positions, or execute in tranches), and notes which option serves which priority. The human can approve immediately rather than reconstructing the analysis.

9.3 Human-in-the-Loop Patterns

Different tasks and risk profiles call for different patterns of human integration. Five patterns span the spectrum from tight oversight to broad autonomy, and selecting the right pattern is itself a design decision with significant consequences.

Approval gates enforce the strictest oversight by separating preparation from authorization. The agent performs all the work—drafting the motion, assembling the trade list, preparing the client communication—but stops short of execution. A human reviews the prepared output and explicitly authorizes the final action. This pattern is essential for irreversible actions where errors cannot be corrected after the fact: court filings, executed trades, sent communications. The cost is latency and human attention; the benefit is a hard guarantee that no consequential action occurs without human review.

Checkpoint reviews insert human validation at natural milestones within a longer workflow. A research agent might present its initial findings and proposed authorities before beginning to draft a memorandum. A due diligence agent might summarize its document review before moving to the analysis phase. These checkpoints prevent the agent from investing significant effort in a direction the human would have redirected, catching misunderstandings early when course correction is cheap.

Confidence-based escalation ties the level of autonomy to the agent's certainty about its outputs. When confidence is high—the answer is clear, the authorities are consistent, the data is unambiguous—

the agent proceeds autonomously. When confidence drops below a threshold, the agent escalates rather than acting on uncertain conclusions. This pattern allows routine cases to flow without human intervention while ensuring that difficult cases receive appropriate attention. The challenge lies in calibrating both the thresholds and the agent's confidence estimates.

Human-as-tool inverts the traditional oversight relationship by treating human expertise as a resource the agent can invoke when needed. Rather than escalating and transferring control, the agent poses a specific question to a human expert, receives the answer, incorporates it into its reasoning, and continues. This pattern works well when the agent needs discrete pieces of human judgment—"Is this clause acceptable under our firm's standards?"—without requiring the human to take over the entire task.

Reversibility classification matches oversight intensity to the stakes of each action. Fully reversible actions—internal drafts, exploratory searches, preliminary analyses—can proceed autonomously because errors can be corrected without consequence. Partially reversible actions warrant checkpoint review. Irreversible actions require approval gates. This framework provides a principled basis for deciding which pattern to apply, grounded in the practical question of what happens if the agent gets it wrong.

9.4 Domain-Specific Escalation Requirements

Beyond general principles, regulated industries impose specific escalation duties grounded in professional responsibility rules and compliance requirements. These are not optional design choices but mandatory constraints that any deployed system must respect.

Legal Practice. Professional responsibility rules create binding escalation requirements for legal agents (American Bar Association 2025; American Bar Association Standing Committee on Ethics and Professional Responsibility 2024). The duty of *competence* under ABA Model Rule 1.1 requires escalation whenever a matter exceeds the agent's training or capabilities—an agent cannot simply do its best on an unfamiliar issue but must recognize its limits and involve qualified counsel. *Privilege protection* demands escalation before any action that might expose attorney-client communications or work product; the consequences of inadvertent disclosure are severe enough that uncertainty should trigger human review. Potential *conflicts of interest*—whether between current clients, with former clients, or arising from the firm's other relationships—must be escalated to counsel for proper conflicts analysis rather than resolved by the agent. And the duty of *candor to the tribunal* requires immediate escalation if the agent discovers adverse authority that may require disclosure; this is not a situation where the agent should exercise judgment about materiality.

Financial Services. Regulatory obligations and fiduciary duties impose parallel requirements on financial agents (Financial Industry Regulatory Authority 2024; Board of Governors of the Federal Reserve System and Office of the Comptroller of the Currency 2011). *Suitability* obligations require

that investment recommendations receive human adviser review before reaching clients, ensuring that a qualified professional has assessed whether the recommendation fits the client's circumstances. Trades approaching *regulatory thresholds*—beneficial ownership reporting triggers, large trader identification requirements, position limits—must be escalated so that compliance personnel can ensure proper filings and approvals. Any encounter with potential *Material Non-Public Information* demands immediate escalation; the agent cannot assess whether information is material or non-public with sufficient reliability, and the consequences of trading on MNPI are catastrophic. And actions that would breach *risk limits*—position concentrations, exposure thresholds, leverage constraints—require escalation for explicit authorization rather than autonomous execution.

9.5 Evaluating Escalation Mechanisms

Escalation evaluation requires six assessments: *coverage* (do all situations that should trigger escalation actually do so?), *calibration* (are thresholds set to balance false positives against missed escalations?), *latency* (does escalation reach humans quickly enough for time-sensitive matters?), *routing* (does it reach the right person with relevant expertise?), *context quality* (can the human decide based on the message alone?), and *response handling* (does the agent correctly incorporate human guidance?). Routing reliability—ensuring escalations reach the right person through channels that cannot be bypassed—is an architectural property; Section 11.5 addresses how escalation hooks must be implemented at the governance layer. See also the consolidated Evaluation Checklist in **??**.

9.6 From Escalation to Delegation

Escalation moves control *upward* in the hierarchy—from agent to human supervisor, from junior to senior, from execution to oversight. But agents can also move control *sideways* by delegating tasks to peer agents with different capabilities. These horizontal relationships introduce coordination challenges distinct from the vertical relationships of escalation.

Critically, escalation and delegation are *orthogonal* design dimensions. A specialist agent can escalate upward to humans while being coordinated sideways by a delegating parent agent—the two patterns do not conflict. However, they must be governed independently: escalation routing must identify the appropriate human supervisor, while delegation must verify that the specialist possesses the required permissions and authority for the delegated task.

Section 10 examines this dimension of agent architecture: how does an agent work with other agents? A coordinating agent might delegate research to a specialist with access to legal databases, drafting to a specialist trained on firm precedents, and financial modeling to a specialist with quantitative capabilities. Each specialist retains its own ability to escalate vertically when it encounters situations requiring human judgment, creating a two-dimensional topology where control flows both upward to humans and sideways to specialists. Understanding both dimensions—escalation and delegation—is essential for designing systems that can handle complex professional workflows while maintaining appropriate human oversight throughout.

10 How Does an Agent Work with Other Agents?

Complex matters require coordination. An M&A partner does not execute everything personally—they coordinate specialists, with corporate counsel reviewing governance, tax specialists analyzing structure, and antitrust counsel assessing regulatory risk. Each specialist contributes deep domain expertise while the partner orchestrates: defining deliverables, integrating work products, and synthesizing conclusions for the client.

A portfolio manager coordinates similarly, with analysts providing company analysis, traders handling execution, risk managers monitoring exposure, and compliance officers verifying adherence. Complex trades require all these perspectives because no single person possesses all necessary expertise.

Agentic systems face the same coordination challenge. A single agent trying to do everything quickly exceeds its competence, permission boundaries, or context limits. Multi-agent architectures mirror professional teams: specialized agents with deep expertise, orchestrators that coordinate them, and structured protocols that enable seamless collaboration.

Delegation

Delegation assigns subtasks from one agent (the coordinator) to another (the specialist). Unlike escalation (agent to human), delegation is agent to agent. The coordinator defines *what* needs to be done; the specialist determines *how*.
Delegation enables parallelization (multiple specialists work simultaneously), specialization (each agent is optimized for its domain), and security isolation (each agent has only the permissions it needs).

10.1 Why Multi-Agent Architectures?

Several factors drive multi-agent designs (Wu et al. 2023; Guo et al. 2024), each reflecting challenges familiar to professional practice.

Specialization allows agents to excel in narrow domains, just as a securities law agent can be optimized for SEC regulations and equipped with EDGAR tools while a tax agent handles tax implications—neither needs expertise in the other's domain. **Security isolation** enforces least privilege: a research agent can read legal databases but cannot file documents, while a filing agent can submit to CM/ECF but cannot access client financial data; if one agent is compromised, damage is contained. **Parallel execution** lets independent workstreams proceed simultaneously, so a document review agent can analyze contracts while a research agent investigates legal issues without either waiting for the other.

Two additional factors favor multi-agent designs in production settings. **Vendor diversity** enables

best-of-breed selection—a specialized legal model handles research, a general model handles drafting, and a fast model handles classification—matching capability to task. **Scale management** addresses context window limits by decomposing tasks across agents, each with focused context, rather than cramming everything into one overwhelmed process.

These benefits come with tradeoffs: coordination overhead increases communication costs, debugging complexity grows when failures span agents, and the attack surface expands with each additional component. The protocols discussed in Section 4.3 and the security controls in Section 5.5 help manage these risks.

10.2 Agent-to-Agent Protocol (A2A)

Just as MCP standardizes how agents access tools (Section 4.3), the Agent-to-Agent Protocol (A2A) standardizes how agents delegate work to each other (Google Developers 2025). A2A uses familiar professional concepts: **Agent Cards** list capabilities (like a specialist's credentials), **Tasks** define delegated work (like engagement letters specifying scope, constraints, and escalation triggers for when specialist judgment reaches authority boundaries), and **Artifacts** are deliverables returned upon completion. The lifecycle mirrors professional delegation: discovery (finding the right specialist), delegation (scoping the work), execution, delivery, and completion. This structure ensures delegated work is scoped, tracked, and auditable.

10.3 Multi-Agent Patterns

Three patterns organize multi-agent collaboration, each with distinct tradeoffs (Wang et al. 2024c).

Sequential Delegation processes work in series: the Coordinator delegates to a Research Agent, whose output flows to an Analysis Agent, then to a Drafting Agent. This pattern is simple to implement and debug—each handoff is clear—but slow, as each stage must wait for the previous one.

Parallel Delegation runs independent workstreams simultaneously. Securities, Tax, and Employment Agents can analyze an acquisition concurrently while the Coordinator integrates findings afterward. This pattern trades coordination complexity for speed, but only works when tasks are truly independent; dependencies between specialists require careful orchestration.

Hierarchical Delegation creates nested authority structures: a Lead Due Diligence Agent delegates to Document Review and Legal Research sub-agents, who may further delegate to specialized tools. This pattern enables deep task decomposition for complex matters but introduces orchestration overhead and debugging challenges when failures occur deep in the hierarchy. Hierarchical delegation mirrors the hierarchical planning pattern in Section 7.1, where decomposition begins with a high-level goal and recursively breaks it into sub-goals. The key difference: planning structures may be internal reasoning steps within a single agent, while delegation structures explicitly distribute work across distinct agent boundaries with attendant coordination and audit requirements.

Figure 7 illustrates these three approaches. Most production systems blend multiple patterns: parallel

agents handle independent analyses while hierarchical structures decompose complex, tightly-coupled tasks. The choice depends on task structure—independent subtasks favor parallelism, dependent workflows favor sequencing, and complex matters with natural subdivisions favor hierarchy.

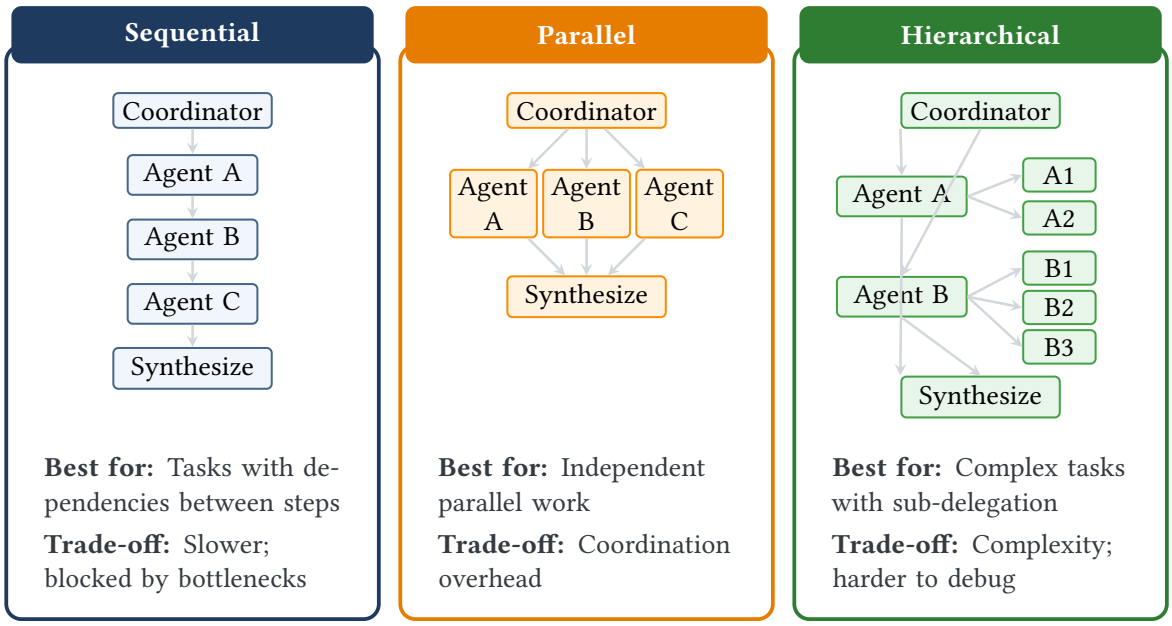


Figure 7: Three multi-agent orchestration patterns. Sequential delegation chains agents in order, ideal for dependent tasks but vulnerable to bottlenecks. Parallel delegation runs agents concurrently, maximizing throughput for independent work but requiring coordination. Hierarchical delegation enables sub-agents to handle specialized sub-tasks, providing flexibility for complex workflows at the cost of debugging complexity.

Consider M&A due diligence as an example. The Orchestrator delegates to specialists in parallel: a Document Processing Agent accesses the data room while a Financial Analysis Agent queries financial databases. Each specialist may use hierarchical delegation internally, with sub-agents handling specific document types or analysis categories. The specialists return structured Artifacts, which the Orchestrator synthesizes into a unified assessment.

10.4 Multi-Agent Workflow Examples

Example: Regulatory Assessment for Financial Product Launch. A fintech company asks about regulatory approvals for a new product—a scenario that spans both legal and financial domains. The Orchestrator decomposes the request across four specialists working in parallel. A **Securities Agent** analyzes SEC guidance and identifies potential registration requirements. A **Banking Agent** checks OCC and FDIC rules, flagging money transmitter licensing needs. A **Consumer Agent** reviews CFPB regulations and highlights disclosure requirements. An **AML Agent** analyzes Bank Secrecy Act obligations and determines KYC needs. Each specialist accesses relevant regulatory databases through the tool integrations discussed in Section 4.3. The Orchestrator synthesizes these parallel findings into a prioritized regulatory roadmap, identifying which approvals are prerequisites

for others, and escalates if any specialist encounters problems requiring human judgment (see Section 9).

10.5 Multi-Agent Risks

Multi-agent systems introduce failure modes beyond single-agent systems (Cemri et al. 2025). **Coordination failures** include deadlock (agents waiting cyclically), divergence (incompatible conclusions requiring reconciliation), and cascading errors (incorrect output propagating through the system). Prevention requires timeouts, validation at handoffs, and clear escalation paths.

Security risks require the same rigor applied to human teams: verifiable agent identities, access policies restricting delegation authority, ethical walls enforced across agent boundaries (an agent on one side of a transaction cannot delegate to specialists with access to the other side's confidential information), and complete delegation logging for audit (OpenID Foundation AI Identity Management Community Group 2024). Comprehensive logging of delegation chains—who delegated what to whom, what constraints were communicated, and what artifacts were returned—is foundational to audit and accountability; Section 11.1 develops the architectural requirements.

10.6 Protocol Selection Guidance

Table 6 summarizes how to choose between MCP and A2A based on task characteristics.

Table 6: Selecting between MCP and A2A based on task characteristics

Task Characteristic	Protocol	Typical Duration	Examples
Immediate, well-defined operation	MCP	Seconds	Query database; retrieve document; run calculation
Delegated work requiring judgment	A2A	Minutes to hours	Assign research; request analysis; coordinate specialists
End-to-end workflow combining both	MCP + A2A	Varies	Due diligence; portfolio rebalancing; regulatory assessment

The decision rule is straightforward: use MCP for well-defined, auditable operations ("fetch this filing," "calculate this metric"), A2A for delegated work requiring specialist judgment ("research and synthesize," "draft and revise," "coordinate across constraints"), and both together for complex workflows. As of late 2025, MCP is production-ready ("Model Context Protocol Specification" 2025), while A2A is maturing with variable cross-vendor reliability ("Agent2Agent Protocol Specification" 2025). Until A2A standardization solidifies, design systems with fallbacks to human coordination for critical decisions.

10.7 From Delegation to Governance

Delegation distributes work, creating governance challenges. Accountability becomes complex: does responsibility lie with the coordinator, the specialist, or the approving human? Information barriers must apply to agents just as they do to humans. Audit trails must span the entire delegation tree.

Section 11 addresses the architectural patterns that enable governance—logging that spans delegation trees, override mechanisms that work across agent boundaries, and privilege enforcement that contains failures. *Governing Agents* then develops the policy layer: delegation authorization, identity management, retention requirements, and organizational accountability.

11 How Do We Design Systems That Can Be Governed?

You cannot audit what you did not log. You cannot enforce privilege boundaries that were never implemented. You cannot override a system that lacks pause mechanisms. When a regulator asks how the compliance agent detected a breach, or when opposing counsel demands the agent's reasoning, architecture determines whether you can answer.

This distinction matters: *governance policy* defines what controls are required; *governance-aware architecture* provides the technical foundation that makes those controls possible. Our next chapter, *Governing Agents*, addresses policy—the five-layer regulatory framework, dimensional calibration, and organizational accountability. This section addresses architecture: the patterns that enable governance, drawn from the capabilities developed throughout this chapter.

Architecture Before Policy

Governance cannot be retrofitted. A system designed without audit logging cannot suddenly produce compliance reports. A system without approval gates cannot enforce human-in-the-loop oversight. A system without state snapshots cannot support rollback or investigation. The architectural decisions in this chapter are not merely technical choices—they are the infrastructure that makes governance possible.

Five architectural patterns enable governance: logging, override mechanisms, state management, privilege enforcement, and escalation hooks. Each pattern draws on capabilities introduced in earlier sections; this section synthesizes them into a coherent governance foundation.

11.1 Logging Architecture

Comprehensive logging is the foundation of all governance. Without logs, there is no audit trail, no incident investigation, no compliance demonstration, and no accountability. Every architectural component in this chapter generates events that must be captured.

Trigger logs record what initiated each agent activation (Section 2). The five evaluation criteria for triggers—coverage, latency, reliability, priority, and *auditability*—require that every trigger event be logged with its source, timestamp, and routing decision. When a regulatory filing triggers a compliance review, the log must capture which filing, when it arrived, and why it was routed to that particular workflow.

Intent logs capture how the agent understood what was asked (Section 3). Goal extraction, constraint identification, and any clarification interactions must be logged to reconstruct what the agent believed it was trying to accomplish. When results are challenged, intent logs reveal whether the agent misunderstood the request, extracted incorrect constraints, or correctly understood but failed in execution.

Perception logs record what information the agent accessed (Section 4.7). Every database query, document retrieval, and API call must be logged with parameters and results. For regulated applications, you must be able to reconstruct what information the agent considered when making a decision. If the agent reviewed ten documents but cited only three, the perception log should show all ten.

Reasoning traces capture how the agent planned and reasoned (Section 7.1). ReAct-style patterns explicitly generate thought-action-observation sequences; these traces make decisions transparent and auditable. For dynamic orchestration, where the agent selects decomposition strategies at runtime, logs must capture the *reasoning* behind orchestration decisions, not just the decisions themselves.

Action logs record what the agent did (Section 5). Every action must be logged with the agent identity, timestamp, parameters, and matter/client context. Action logs enable reconstruction: given a complaint about an erroneous filing, you can trace back through the action log to find when it was filed, what parameters were used, and what reasoning preceded it.

Memory access logs track reads and writes to persistent state (Section 6). Each operation should be logged with a timestamp, the requesting agent or user, and the matter identifier. These logs support compliance review, breach investigation, and documentation that appropriate controls were maintained.

Delegation logs capture agent-to-agent interactions (Section 10). When a coordinator delegates to specialists, the log must record the delegation contract, the specialist's identity, and the returned artifacts. Complete delegation logging enables audit trails that span the entire delegation tree.

Structured Logging Schema

Effective governance requires structured logs, not free-form text. A minimal schema includes:

- **Timestamp**: ISO 8601 with timezone
- **Event type**: trigger, perception, reasoning, action, memory, delegation
- **Agent identity**: which agent or component generated the event
- **Matter/client context**: isolation identifier for multi-tenant systems
- **Parameters**: structured data specific to the event type
- **Outcome**: success, failure, or pending

Structured logs enable programmatic analysis, anomaly detection, and automated compliance reporting.

11.2 Override and Circuit Breaker Patterns

Governance requires the ability to intervene. Humans must be able to pause, redirect, or stop agent execution at any point. Three patterns provide this capability.

Circuit breakers automatically halt execution when anomalies are detected (Section 5.7). These include failure count thresholds (e.g., stop after three consecutive failures), spike detection (e.g., pause if action rate exceeds 5× baseline), and error rate monitoring. Circuit breakers transform runaway failures into controlled pauses, buying time for human intervention. The key architectural requirement is that circuit breaker state must be external to the agent—an agent cannot be trusted to halt itself reliably.

Step limits provide hard backstops (Section 8, specifically Section 8.4). Regardless of other conditions, after N total steps the agent must stop and require human approval before continuing. This prevents unbounded execution even when other detection mechanisms fail. The appropriate limit depends on the task, but the key is that *some* limit exists and is enforced by the orchestration layer, not by the agent itself.

Manual override interfaces allow humans to intervene at any time. This requires:

- **Pause**: Suspend execution while preserving state, allowing inspection before resumption.
- **Abort**: Terminate execution immediately, triggering graceful degradation (Section 8.6) to preserve partial work.
- **Redirect**: Modify the agent's goal or constraints mid-execution without losing accumulated context.

These interfaces must be accessible to authorized personnel regardless of the agent's internal state. An agent stuck in a loop or processing a high-volume task must still respond to override commands. This typically requires a separate control plane—a monitoring service that can signal the agent through a channel the agent cannot ignore.

The Red Button Problem

Every production agentic system needs an emergency stop mechanism—the "red button" that immediately halts all agent activity. This is not merely a graceful shutdown request; it is an immediate cessation of all external actions. Implementing this requires that action tools check a global halt flag before execution, that the halt flag is stored externally (not in agent memory), and that the flag can be set through an out-of-band channel that bypasses normal request processing.

11.3 State Snapshots and Checkpoints

Three governance requirements depend on state management: reproducibility (reconstructing what happened), rollback (undoing problematic changes), and resumability (continuing after interruption).

Reproducibility requires capturing sufficient state to replay decisions. When a regulator asks why the agent made a particular recommendation, you must be able to reconstruct the agent's context at that moment: what was in memory, what had been retrieved, what reasoning had occurred. This is more demanding than simple logging—it requires periodic state snapshots that capture the complete agent context, not just individual events.

Rollback capability enables recovery from errors (Section 5.3). For reversible actions, rollback is straightforward: delete the draft, revert the database change. For partially reversible actions, checkpoints enable recovery to the last known good state. The architecture must classify actions by reversibility and maintain appropriate checkpoints for recovery.

Checkpoint reviews support human-in-the-loop oversight (Section 9.3). At natural milestones within longer workflows, the agent can present its current state and proposed next steps for human validation. These checkpoints prevent the agent from investing significant effort in a wrong direction. Architecturally, checkpoints require the ability to serialize agent state, present it in human-readable form, and resume from exactly that point after approval.

Graceful degradation preserves value when execution terminates early (Section 8.6). Whether due to budget exhaustion, confidence drops, or manual abort, the agent should deliver whatever value it has accumulated. This requires maintaining partial results throughout execution, not just at the end. If interrupted after reviewing fifty of one hundred documents, the agent should be able to report findings from the fifty it completed.

Memory state deserves special attention (Section 6). When behavior depends on accumulated memory, audit trails must capture not just inputs and outputs but the memory state at key decision points. Which version of the agent produced this output? What was in its memory at the time? Without memory snapshots, accountability for adaptive agents becomes impossible.

11.4 Least Privilege Enforcement

Least privilege—granting only the minimum permissions required for a task—limits blast radius when things go wrong. This principle appears throughout the chapter; governance-aware architecture enforces it systematically.

Tool-level permissions control what actions an agent can take (Section 5.5). Role-based access control (RBAC) restricts which agents can invoke which tools. A research agent should not have filing permissions; a filing agent should not have broad database access. The MCP architecture supports this through tool manifests that declare required permissions, allowing the host to enforce access control.

Resource-level permissions control what data an agent can access (Section 4.3). MCP Resources are explicitly read-only, separating perception from action. An agent with resource access can retrieve documents but cannot modify or file them. This separation enables fine-grained access control that mirrors how organizations already think about permissions.

Namespace isolation enforces boundaries in memory systems (Section 6.4). Matter isolation ensures that information from one client engagement cannot leak into another. Role-based permissions determine which agents can access each namespace. These controls mirror the ethical wall policies that law firms and financial institutions maintain for human professionals.

Multi-agent isolation uses architectural boundaries to enforce privilege separation (Section 10.1). Rather than giving one agent broad permissions, decompose work across specialists, each with narrow permissions. A research agent can read legal databases but cannot file documents; a filing agent can submit to CM/ECF but cannot access client financial data. If one agent is compromised, damage is contained.

Privilege Escalation via Tool Chaining

A sophisticated threat involves combining multiple low-privilege tools to achieve a high-privilege outcome (Section 5.5). An agent with "read email" and "send email" permissions might forward confidential information to an external address. Mitigation requires analyzing tool combinations, not just individual tools, and requiring approval for sequences that cross security boundaries.

11.5 Escalation Hooks

Escalation transfers control from agent to human when autonomous execution should stop (Section 9). Governance-aware architecture must define clear escalation interfaces that trigger reliably under specified conditions.

Confidence-based escalation triggers when the agent's uncertainty exceeds acceptable thresholds (Section 9.1). The architecture must support confidence estimation and threshold configuration.

Different tasks may require different thresholds: a routine lookup tolerates higher uncertainty than a regulatory filing.

Authority-based escalation triggers when a proposed action exceeds the agent's delegated authority. Binding commitments, expenditures above a threshold, or communications with external parties may all require human approval. The architecture must support authority boundaries that can be configured per agent, per task, or per action type.

Budget-based escalation triggers when resource consumption approaches limits (Section 7.4). Rather than simply halting at budget exhaustion, the agent can escalate when 80% of budget is consumed, giving humans the opportunity to extend resources or redirect effort. This requires the budget monitoring system to support threshold alerts, not just hard limits.

Anomaly-based escalation triggers when the agent detects unusual patterns—repeated failures, unexpected results, or behavior outside historical norms. This complements circuit breakers: where circuit breakers halt execution, anomaly escalation requests human judgment about whether to continue.

For all escalation types, the architecture must ensure that escalation requests reach humans reliably. An agent that escalates to an unmonitored queue has not meaningfully escalated. Escalation routing, notification, and tracking are architectural requirements, not operational afterthoughts.

11.6 The Governance Surface

Together, these five patterns define the **governance surface**—the interface through which governance systems interact with the agent. Table 7 summarizes the architectural requirements.

Table 7: Governance surface requirements by capability

Capability	Requirement
Logging	Structured events from every component
Override	Pause, abort, and redirect controls
State Management	Checkpoints with rollback support
Least Privilege	RBAC at tool, resource, and namespace levels
Escalation Hooks	Configurable triggers with reliable routing

This governance surface is what *Governing Agents* builds upon. The five-layer regulatory framework (foundational law, professional obligations, sector regulation, AI-specific regulation, voluntary frameworks) imposes requirements; the governance surface provides the technical means to satisfy them. Without logging, there is no audit trail for regulators. Without override capability, there is no human oversight mechanism. Without state management, there is no reproducibility for investigations. Without least privilege, there is no containment of failures. Without escalation hooks, there is no transfer of control to humans.

The architectural decisions made throughout this chapter—how you design triggers, intent extraction, perception tools, action controls, memory systems, planning mechanisms, termination conditions, escalation protocols, and delegation patterns—collectively determine whether your system can be governed. This section has addressed *how* architecture enables governance; *Governing Agents* addresses *what* governance requires and how to calibrate controls to your specific context and risk profile.

12 Conclusion

Understanding agent architecture does not require becoming a developer. The goal is meaningful participation in decisions that affect your practice, your clients, and your professional responsibilities.

With architectural literacy, you can evaluate vendor claims with precision. When demonstrations look impressive, you know what questions to ask: What triggered the run? How was intent validated? What data sources were accessed? What approval gates exist? How is isolation enforced? What happens when confidence drops? Impressive outputs do not guarantee sound architecture; architectural literacy lets you probe beneath the surface.

You can specify requirements in terms that technical teams understand. Rather than vague requests for "AI that helps with research," you can describe what you need: perception tools for specific databases, action controls with appropriate approval gates, memory systems with client isolation, and escalation triggers for low-confidence situations. Shared vocabulary bridges the gap between professional requirements and technical implementation.

You can demand governance artifacts, not governance promises. If a vendor cannot demonstrate what the agent accessed, what it did, and why it stopped, the system is not ready for regulated practice. The governance surface requirements from Section 11.6—structured logging, override mechanisms, state snapshots, least privilege enforcement, reliable escalation—are your checklist for what must be demonstrable before deployment.

And you can participate meaningfully in governance design. Governance policy depends on architectural infrastructure; controls must be built in rather than bolted on, and design choices determine what oversight is possible.

12.1 Tradeoffs Require Judgment

Every architectural capability involves tradeoffs. Richer memory improves context but increases latency and cost. Aggressive escalation improves safety but reduces autonomy and throughput. Tighter approval gates reduce risk but slow execution. More sophisticated planning handles complexity but consumes more resources and introduces more potential failure points. There are no universally correct answers—only choices that must be calibrated to your context, your risk tolerance, and your

professional obligations.

Current limitations make this calibration essential. Today's agents perform well on constrained, well-defined tasks but struggle as complexity and duration increase. The "reliability cliff" discussed in Section 8.5 is real: success rates tend to degrade as tasks grow longer and more open-ended. Rather than avoiding agentic systems, these limitations argue for scoping them appropriately, adding checkpoints, and designing for human oversight.

For now, agents are best treated as capable assistants that amplify human judgment. The value lies in augmentation: agents handle the routine so professionals can focus on the consequential, with controls that keep humans in the loop where judgment matters.

As technology improves, the tradeoffs will shift. But the ten architectural questions will remain, and the framework you have learned will stay useful even as implementations evolve.

12.2 From Architecture to Governance

This chapter has addressed how to design agents. The natural next question is how to govern them.

Architecture enables governance without determining governance. The governance surface provides the technical means—logging, overrides, state management, privileges, escalation—but policy determines how those means are used. How should approval thresholds be set? Who is accountable when an agent errs? What documentation must be maintained? How do professional duties translate into system requirements? Policy, regulation, and professional responsibility must answer these questions.

Governing Agents, the next chapter in this series, addresses these questions directly. It provides a five-layer regulatory framework spanning foundational law, professional obligations, sector-specific regulation, AI-specific requirements, and voluntary frameworks. It offers a dimensional approach to calibrating controls rather than one-size-fits-all checklists, grounded in the architectural understanding you have now developed.

You are equipped to engage with those governance frameworks because you understand what the architecture can and cannot make observable and enforceable: what logging captures and what it misses, what escalation protocols can enforce and what they cannot, and what memory isolation protects and what additional controls it requires. Governance sits atop the architectural foundation this chapter has built.

12.3 Agents, Understood

Agents are not magic. They are triggers and channels, intent extraction and constraint validation, perception tools and action controls, memory systems and planning patterns, termination conditions and escalation protocols, delegation architectures and governance surfaces. They are the structural capabilities that any system—human or artificial—requires to do real cognitive work.

These capabilities are now visible to you. You can see past an interface to the architecture beneath, evaluate claims with informed skepticism, and specify requirements that bridge professional needs and technical implementation.

Architectural literacy makes you a capable evaluator, a precise specifier, and an informed participant in decisions about technology that affects your practice. As agents become more prevalent in legal and financial work, that literacy becomes increasingly valuable.

You now have the foundation to evaluate, specify, deploy, and govern agentic systems with the same rigor you apply to the professional teams these systems are meant to augment. The work of understanding agents continues in *Governing Agents*, where we address what controls are required, how to calibrate them, and who bears responsibility.

References

Abbasi Yadkori, Yasin, Ilja Kuzborskij, David Stutz, András György, Adam Fisch, Arnaud Doucet, Iuliya Beloshapka, Wei-Hung Weng, Yao-Yuan Yang, Csaba Szepesvári, Ali Taylan Cemgil, and Nenad Tomasev (2024). *Mitigating LLM Hallucinations via Conformal Abstention*. Develops principled procedure for determining when LLMs should abstain (say "I don't know") instead of hallucinating; uses conformal prediction to provide theoretical guarantees on hallucination rates. arXiv: 2405.01563 [cs.LG]. URL: https://arxiv.org/abs/2405.01563 (visited on 12/19/2025).

Agent2Agent Protocol Specification (2025). Official specification for A2A protocol; supports HTTP, SSE, gRPC transports; security card signing added November 2025. URL: https://a2a-protocol.org/latest/ (visited on 11/27/2025).

Allen, James F. and C. Raymond Perrault (1980). "Analyzing Intention in Utterances". In: *Artificial Intelligence* 15.3. Foundational paper on plan-based inference of speaker intentions in cooperative dialogue; establishes framework for understanding how listeners infer goals from utterances, pp. 143–178. DOI: 10.1016/0004-3702(80)90042-9. URL: https://doi.org/10.1016/0004-3702(80)90042-9 (visited on 12/19/2025).

American Bar Association (2025). *Model Rules of Professional Conduct, Rule 1.1: Competence*. Primary authority establishing duty of competence including technology competence (Comment 8 amended 2012). URL: https://www.americanbar.org/groups/professional_responsibility/publications/model_rules_of_professional_conduct/rule_1_1_competence/ (visited on 12/13/2025).

American Bar Association Standing Committee on Ethics and Professional Responsibility (July 2024). *Formal Opinion 512: Generative Artificial Intelligence Tools*. Tech. rep. Addresses ethical obligations when using generative AI; covers competence, confidentiality, supervision, and billing. American Bar Association. URL: https://www.americanbar.org/groups/professional_responsibility/publications/ethics_opinions/formal-opinion-512/ (visited on 11/27/2025).

Anthropic (Nov. 2024). *Introducing the Model Context Protocol*. Open standard for connecting AI systems to data sources; pre-built servers for Google Drive, Slack, GitHub, Postgres; SDKs for Python, TypeScript, C#. URL: https://www.anthropic.com/news/model-context-protocol (visited on 11/27/2025).

Austin, J. L. (1962). *How to Do Things with Words*. Posthumously published William James Lectures (Harvard, 1955) establishing speech act theory; introduces performative utterances—utterances that constitute actions rather than describe them (e.g., "I promise," "I sentence you"). Second edition edited by J. O. Urmson and Marina Sbisà, Harvard University Press, 1975, ISBN 978-0-674-41152-4. Oxford: Oxford University Press.

Board of Governors of the Federal Reserve System and Office of the Comptroller of the Currency (Apr. 2011). *Supervisory Guidance on Model Risk Management*. Tech. rep. SR Letter 11-7 / OCC Bulletin

2011-12. Foundational guidance on model risk management for financial institutions; defines model risk, validation requirements, and governance standards; increasingly relevant to AI/ML models. Federal Reserve Board. URL: https://www.federalreserve.gov/supervisionreg/srletters/sr1107.htm (visited on 12/13/2025).

Bommarito, Michael James, Jillian Bommarito, and Daniel Martin Katz (Nov. 2025). *What is an Agent? A Conceptual Primer and History of Agents and Agentic AI*. Working Paper 5806982. Part I of the Agents series from *Artificial Intelligence for Law and Finance*; establishes GPA+IAT framework synthesizing seven decades of agency scholarship from philosophy, psychology, law, economics, and computer science. SSRN. DOI: 10.2139/ssrn.5806982. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5806982 (visited on 12/14/2025).

Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei (2020). "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. Seminal paper introducing GPT-3 (175B parameters) and establishing in-context learning: the ability of large language models to adapt to new tasks from examples provided in the prompt without parameter updates, pp. 1877–1901. URL: https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html (visited on 12/19/2025).

Cemri, Mert, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica (2025). *Why Do Multi-Agent LLM Systems Fail?* Introduces MAST taxonomy with 14 failure modes in 3 categories (system design, inter-agent misalignment, task verification); analyzes 1600+ annotated traces across 7 MAS frameworks; high inter-annotator agreement ($\kappa = 0.88$). arXiv: 2503.13657 [cs.AI]. URL: https://arxiv.org/abs/2503.13657 (visited on 12/19/2025).

Clop, Cody and Yannick Teglia (2024). *Backdoored Retrievers for Prompt Injection Attacks on Retrieval Augmented Generation of Large Language Models*. Demonstrates corpus poisoning and backdoor attacks on RAG systems; achieves high attack success rates with minimal poisoned documents; security implications for memory-based agents. arXiv: 2410.14479 [cs.CR]. URL: https://arxiv.org/abs/2410.14479 (visited on 12/19/2025).

Dahl, Matthew, Varun Magesh, Mirac Suzgun, and Daniel E. Ho (2024). "Hallucination-Free? Assessing the Reliability of Leading AI Legal Research Tools". In: *Journal of Empirical Legal Studies*. Finds RAG-based legal tools (Lexis+ AI, Westlaw AI) hallucinate 17–33% of the time; challenges vendor claims of hallucination-free performance. DOI: 10.1111/jels.12394. URL: https://law.stanford.edu/wp-content/uploads/2024/05/Legal_RAG_Hallucinations.pdf (visited on 12/13/2025).

Financial Industry Regulatory Authority (June 2024). *Regulatory Notice 24-09: FINRA Reminds Member Firms of Regulatory Obligations When Using Artificial Intelligence*. Tech. rep. Official guidance establishing supervision and governance requirements for AI in financial services under Rule 3110. FINRA. URL: https://www.finra.org/rules-guidance/notices/24-09 (visited on 12/13/2025).

Gomez, Camilo, Seong Mi Cho, Sue Ke, Chien-Ming Huang, and Mathias Unberath (2025). "Human-AI collaboration is not very collaborative yet: a taxonomy of interaction patterns in AI-assisted decision making from a systematic review". In: *Frontiers in Computer Science* 6. Systematic review of 105 articles developing taxonomy of human-AI interaction patterns; identifies limited bidirectional collaboration in current AI-assisted decision systems, p. 1521066. DOI: 10.3389/fcomp.2024.1521066. URL: https://www.frontiersin.org/journals/computer-science/articles/10.3389/fcomp.2024.1521066/full (visited on 12/19/2025).

Google Developers (Apr. 2025). *Announcing the Agent2Agent Protocol (A2A)*. Open protocol for agent-to-agent communication using JSON-RPC 2.0 over HTTP; donated to Linux Foundation; features Agent Cards for capability discovery; 50+ launch partners. URL: https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/ (visited on 11/27/2025).

Grinsztajn, Nathan, Johan Ferret, Olivier Pietquin, Philippe Preux, and Matthieu Geist (2021). "There Is No Turning Back: A Self-Supervised Approach for Reversibility-Aware Reinforcement Learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 34. Self-supervised approach for learning reversibility in RL; agents learn to distinguish reversible from irreversible actions without explicit labels; foundational for safety-aware autonomous systems, pp. 1898–1911. URL: https://papers.nips.cc/paper_files/paper/2021/hash/0e98aeeb54acf612b9eb4e48a269814c-Abstract.html (visited on 12/19/2025).

Grossman, Maura R. and Gordon V. Cormack (2011). "Technology-Assisted Review in E-Discovery Can Be More Effective and More Efficient Than Exhaustive Manual Review". In: *Richmond Journal of Law & Technology* 17.3, 11. Foundational paper establishing precision, recall, and F1 as standard metrics for e-discovery effectiveness; cited in Da Silva Moore v. Publicis Groupe and subsequent TAR case law. URL: https://scholarship.richmond.edu/jolt/vol17/iss3/5/ (visited on 12/19/2025).

Guo, Taicheng, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang (2024). "Large Language Model Based Multi-agents: A Survey of Progress and Challenges". In: *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24)*. Comprehensive survey on LLM-based multi-agent systems covering architectures, coordination mechanisms, and problem-solving capabilities. URL: https://arxiv.org/abs/2402.01680 (visited on 12/13/2025).

Hollan, James, Edwin Hutchins, and David Kirsh (2000). "Distributed Cognition: Toward a New Foundation for Human-Computer Interaction Research". In: *ACM Transactions on Computer-Human Interaction* 7.2. Seminal paper proposing distributed cognition as framework for HCI; cognition

extends beyond individuals to include artifacts, environment, and social structures, pp. 174–196. DOI: 10.1145/353485.353487. URL: https://doi.org/10.1145/353485.353487 (visited on 12/19/2025).

Hollnagel, Erik and David D. Woods (2005). *Joint Cognitive Systems: Foundations of Cognitive Systems Engineering*. Foundational text establishing cognitive systems engineering framework; introduces joint cognitive systems as units of analysis for human-technology interaction. Boca Raton, FL: CRC Press. DOI: 10.1201/9781420038194. URL: https://doi.org/10.1201/9781420038194 (visited on 12/19/2025).

Kadavath, Saurav, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, et al. (2022). *Language Models (Mostly) Know What They Know*. Anthropic study on LLM confidence calibration; finds models can predict their own accuracy but require careful prompting; relevant to confidence thresholds for agent escalation. arXiv: 2207.05221 [cs.CL]. URL: https://arxiv.org/abs/2207.05221 (visited on 12/13/2025).

Kim, Sehoon, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami (2024). "An LLM Compiler for Parallel Function Calling". In: *International Conference on Machine Learning (ICML)*. Compiler-inspired architecture for parallel function execution in agents; demonstrates up to 3.7x latency speedup and 6.7x cost savings vs. ReAct. URL: https://arxiv.org/abs/2312.04511 (visited on 12/13/2025).

Legal Information Institute (2024). *Rule 12. Defenses and Objections: When and How Presented*. Federal rule establishing 21-day deadline for responsive pleadings; critical for deadline calculation in litigation agents. URL: https://www.law.cornell.edu/rules/frcp/rule_12 (visited on 12/13/2025).

Lewis, Patrick, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela (2020). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. Original RAG paper introducing the paradigm of combining parametric (LLM) and non-parametric (retrieval) memory for generation tasks, pp. 9459–9474. URL: https://arxiv.org/abs/2005.11401 (visited on 12/13/2025).

Liu, Yupei, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong (2024). "Formalizing and Benchmarking Prompt Injection Attacks and Defenses". In: *Proceedings of the 33rd USENIX Security Symposium*. First formal framework for prompt injection attacks; comprehensive benchmark of attack vectors and defense mechanisms; peer-reviewed security venue. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/liu-yupei (visited on 12/19/2025).

Ma, Chang, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He (2024). "AgentBoard: An Analytical Evaluation Board of Multi-turn LLM Agents". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 37. NeurIPS 2024 Oral; introduces fine-grained progress rate metric that captures incremental advancements in

multi-turn agent tasks. arXiv: 2401.13178. URL: https://arxiv.org/abs/2401.13178 (visited on 12/19/2025).

Ma, Xinbei, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan (2023). "Query Rewriting for Retrieval-Augmented Large Language Models". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Introduces Rewrite-Retrieve-Read framework; uses trainable rewriter to transform queries for better retrieval; demonstrates improved QA accuracy through query adaptation. Association for Computational Linguistics. URL: https://aclanthology.org/2023.emnlp-main.322/ (visited on 12/19/2025).

Madras, David, Toniann Pitassi, and Richard Zemel (2018). "Predict Responsibly: Improving Fairness and Accuracy by Learning to Defer". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 31. Foundational work on learning when AI should defer to human decision-makers; frames deferral as joint optimization of accuracy and fairness; basis for confidence-based escalation. URL: https://proceedings.neurips.cc/paper/2018/hash/09d37c08f7b129e96277388757530c72-Abstract.html (visited on 12/19/2025).

McCarthy, John and Patrick J. Hayes (1969). "Some Philosophical Problems from the Standpoint of Artificial Intelligence". In: *Machine Intelligence 4*. Ed. by B. Meltzer and D. Michie. Foundational paper introducing the frame problem in AI: when an action occurs, what changes and what stays the same? Also introduces situation calculus for reasoning about actions and their effects. Edinburgh: Edinburgh University Press, pp. 463–502. URL: https://www-formal.stanford.edu/jmc/mcchay69.pdf (visited on 12/19/2025).

METR (Mar. 2025). *Measuring AI Ability to Complete Long Tasks*. Empirical study finding AI agent success rates inversely correlated to task duration; 100% success on tasks under 4 minutes, under 10% for tasks over 4 hours; capability doubling time approximately 7 months. URL: https://metr.org/blog/2025-03-19-measuring-ai-ability-to-complete-long-tasks/ (visited on 11/27/2025).

Mo, Guozhao, Wenliang Zhong, Jiawei Chen, Xuanang Chen, Yaojie Lu, Hongyu Lin, Ben He, Xianpei Han, and Le Sun (2025). "LiveMCPBench: Can Agents Navigate an Ocean of MCP Tools?" In: *arXiv preprint arXiv:2508.01780*. Benchmark evaluating agent performance over large MCP tool ecosystems; discusses MCP server ecosystem scale (10,000+ servers). DOI: 10.48550/arXiv.2508.01780. URL: https://arxiv.org/abs/2508.01780 (visited on 12/15/2025).

Model Context Protocol Specification (2025). Official MCP documentation and specification; server and client SDKs; community server directory. URL: https://modelcontextprotocol.io/ (visited on 11/27/2025).

Mosqueira-Rey, Eduardo, Elena Hernández-Pereira, David Alonso-Ríos, José Bobes-Bascarán, and Ángel Fernández-Leal (2023). "Human-in-the-loop machine learning: a state of the art". In: *Artificial Intelligence Review* 56. Comprehensive survey of HITL-ML covering active learning, interactive ML, machine teaching, curriculum learning, and explainable AI; includes confidence-based escalation approaches, pp. 3005–3054. DOI: 10.1007/s10462-022-10246-w. URL: https://link.springer.com/article/10.1007/s10462-022-10246-w (visited on 12/19/2025).

Nielsen, Jakob (June 2023). *AI: First New UI Paradigm in 60 Years*. Introduces concept of intent-based outcome specification as the third UI paradigm after batch processing and command-based interaction; argues AI shifts locus of control from user specifying commands to user specifying desired outcomes. URL: https://www.nngroup.com/articles/ai-paradigm/ (visited on 12/13/2025).

OpenID Foundation AI Identity Management Community Group (2024). "Identity Management for Agentic AI: The New Frontier of Authorization, Authentication, and Security for an AI Agent World". In: *arXiv preprint arXiv:2510.25819*. Addresses authentication, authorization, and identity management in multi-agent systems; proposes DIDs, Verifiable Credentials, and Zero Trust principles. URL: https://arxiv.org/abs/2510.25819 (visited on 12/13/2025).

Ouyang, Long, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe (2022). *Training Language Models to Follow Instructions with Human Feedback*. Introduces InstructGPT and RLHF methodology; explicitly frames alignment as following user intent across diverse tasks; foundational for modern instruction-following LLMs. arXiv: 2203.02155 [cs.CL]. URL: https://arxiv.org/abs/2203.02155 (visited on 12/19/2025).

OWASP Foundation (2025). *OWASP Top 10 for Large Language Model Applications*. Security vulnerabilities in LLM applications; ranks prompt injection as critical vulnerability #1. URL: https://owasp.org/www-project-top-10-for-large-language-model-applications/ (visited on 11/27/2025).

Parasuraman, Raja, Thomas B. Sheridan, and Christopher D. Wickens (2000). "A Model for Types and Levels of Human Interaction with Automation". In: *IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans* 30.3. Foundational framework defining 10 levels of automation across four information-processing stages; canonical reference for human-automation interaction and approval gate design, pp. 286–297. DOI: 10.1109/3468.844354. URL: https://ieeexplore.ieee.org/document/844354 (visited on 12/19/2025).

Park, Joon Sung, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein (2023). "Generative Agents: Interactive Simulacra of Human Behavior". In: *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*. Introduces memory stream architecture with reflection for long-term agent behavior; foundational for episodic memory and learning in agent systems. ACM. DOI: 10.1145/3586183.3606763. URL: https://arxiv.org/abs/2304.03442 (visited on 12/19/2025).

Press, Ofir, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis (2023). "Measuring and Narrowing the Compositionality Gap in Language Models". In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. Quantifies compositionality gap where models correctly solve sub-problems but fail on overall compositional tasks; demonstrates performance degradation as composition steps increase; gap remains constant at 40% across model sizes.

Singapore: Association for Computational Linguistics, pp. 5687–5711. arXiv: 2210.03350. URL: https://arxiv.org/abs/2210.03350 (visited on 12/19/2025).

Rao, Anand S. and Michael P. Georgeff (1995). "BDI Agents: From Theory to Practice". In: *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*. Foundational work on Belief-Desire-Intention cognitive architecture for agents. AAAI Press, pp. 312–319. URL: https://cdn.aaai.org/ICMAS/1995/ICMAS95-042.pdf (visited on 12/13/2025).

Robertson, Stephen and Hugo Zaragoza (2009). "The Probabilistic Relevance Framework: BM25 and Beyond". In: *Foundations and Trends in Information Retrieval*. Vol. 3. 4. Comprehensive review of BM25 probabilistic ranking algorithm, widely used for keyword search in hybrid RAG retrieval. Now Publishers, pp. 333–389. DOI: 10.1561/1500000019. (Visited on 12/13/2025).

Russell, Stuart and Peter Norvig (2020). *Artificial Intelligence: A Modern Approach*. 4th ed. Foundational textbook establishing agent architecture: perception-action loop, rational agents, and agent type taxonomy (simple reflex, model-based reflex, goal-based, utility-based). Pearson. URL: https://aima.cs.berkeley.edu/ (visited on 12/13/2025).

Wang, Lei, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen (2024a). "A Survey on Large Language Model based Autonomous Agents". In: *Frontiers of Computer Science* 18.6. Peer-reviewed survey proposing unified framework for LLM-based agents covering profile, memory, planning, and action modules, p. 186345. DOI: 10.1007/s11704-024-40231-1. URL: https://link.springer.com/article/10.1007/s11704-024-40231-1 (visited on 12/19/2025).

Wang, Wenxuan, Juluan Shi, Zixuan Ling, Yuk-Kit Chan, Chaozheng Wang, Cheryl Lee, Youliang Yuan, Jen-tse Huang, Wenxiang Jiao, and Michael R. Lyu (2024b). *Learning to Ask: When LLM Agents Meet Unclear Instruction*. Proposes Ask-when-Needed (AwN) framework prompting LLMs to ask questions when encountering unclear instructions; creates NoisyToolBench benchmark for imperfect instructions. arXiv: 2409.00557 [cs.CL]. URL: https://arxiv.org/abs/2409.00557 (visited on 12/13/2025).

Wang, Yaoxiang, Zhiyong Wu, Junfeng Yao, and Jinsong Su (2024c). "TDAG: A Multi-Agent Framework based on Dynamic Task Decomposition and Agent Generation". In: *Neural Networks*. Addresses dynamic task decomposition in multi-agent systems; demonstrates complex task breakdown with specialized subagents. URL: https://arxiv.org/abs/2402.10178 (visited on 12/13/2025).

Wu, Qingyun, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang (2023). "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation". In: *arXiv preprint arXiv:2308.08155*. Foundational framework for multi-agent conversation systems; covers customizable agents, flexible conversation patterns. URL: https://arxiv.org/abs/2308.08155 (visited on 12/13/2025).

Xu, Binfeng, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu (2023). *ReWOO: Decoupling Reasoning from Observations for Efficient Augmented Language Models*. Modular paradigm separating reasoning from tool observations; achieves 5x token efficiency

and 4% accuracy improvement over ReAct on HotpotQA. arXiv: 2305.18323 [cs.CL]. URL: https://arxiv.org/abs/2305.18323 (visited on 12/13/2025).

Yao, Shunyu, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao (2022). "ReAct: Synergizing Reasoning and Acting in Language Models". In: *arXiv preprint arXiv:2210.03629*. Introduces ReAct pattern: alternating reasoning and acting in language models. URL: https://arxiv.org/abs/2210.03629 (visited on 12/19/2025).

Yu, Yue, Wei Ping, Zihan Liu, Boxin Wang, Jiaxuan You, Chao Zhang, Mohammad Shoeybi, and Bryan Catanzaro (2024). *RankRAG: Unifying Context Ranking with Retrieval-Augmented Generation in LLMs*. Instruction-tunes single LLM for both context ranking and answer generation; outperforms GPT-4 on knowledge-intensive benchmarks; demonstrates reranking integrated into generation. arXiv: 2407.02485 [cs.CL]. URL: https://arxiv.org/abs/2407.02485 (visited on 12/19/2025).

Zhang, Michael J. Q., W. Bradley Knox, and Eunsol Choi (2024). *Modeling Future Conversation Turns to Teach LLMs to Ask Clarifying Questions*. Finds LLMs often respond by presupposing a single interpretation of ambiguous requests rather than asking clarifying questions; proposes preference labeling using simulated future turns to teach when clarification is needed. arXiv: 2410.13788 [cs.CL]. URL: https://arxiv.org/abs/2410.13788 (visited on 12/13/2025).

Zou, Andy, Long Phan, Justin Wang, Derek Duenas, Maxwell Lin, Maksym Andriushchenko, Rowan Wang, Zico Kolter, Matt Fredrikson, and Dan Hendrycks (2024). "Improving Alignment and Robustness with Circuit Breakers". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 37. Introduces circuit breakers as interrupt mechanisms for AI agents; demonstrates reductions in harmful actions under attack; extends to agent function-calling safety. arXiv: 2406.04313. URL: https://arxiv.org/abs/2406.04313 (visited on 12/19/2025).