Agents

Part II: How to Build an Agent

Architectures, Protocols, and Technical Evaluation

Michael J Bommarito II · Daniel Martin Katz · Jillian Bommarito

December 13, 2025

Contents

1 Introduction

The previous chapter answered: *What is an agent?* This chapter answers: *How do you build one?*

Moving from "what is an agent?" to "how do you build one?" changes what you need to know. The GPA+IAT framework from the previous chapter—Goals, Perception, Action, Iteration, Adaptation, Termination—gives you the vocabulary to recognize agentic systems and evaluate whether something truly qualifies as an agent. But recognizing a system is not the same as deploying it, governing it, or knowing when a vendor is overselling it. For that, you need to understand how agents are actually built: what architectural decisions shape capabilities, what tradeoffs those decisions create, and where things go wrong.

This chapter provides that understanding through ten questions—the questions you should ask when evaluating, deploying, or governing any agentic system. Each question maps to a capability every useful agent needs, each capability involves design choices with real consequences, and those choices determine not just what the system can do but how reliably it performs, how it fails, and what controls are possible.

The Core Insight

Agents are not magic; they are architecture.
Every capability that makes an agent useful—understanding what you asked, finding relevant information, taking action, remembering context, planning complex work, knowing when to stop, asking for help when stuck—corresponds to a concrete design decision. Those decisions have tradeoffs. Those tradeoffs have consequences.
You do not need to build agents yourself. But you need to know what questions to ask—because architectural decisions made at design time determine what the system can do, how reliably it performs, what can go wrong, and what controls are possible.

1.1 Agents as Professional Teams

The ten questions in this chapter are not arbitrary. They emerge from what agentic systems are designed to do: augment human professionals, automate routine workflows, or eventually replace entire organizational functions. Whatever the ambition, the system must handle the same work those humans and organizations currently perform. A legal research agent must do what a research associate does. A portfolio monitoring agent must do what an analyst does. The capabilities required are not determined by the technology; they are determined by the work. And the work has structure that any system—human or artificial—must accommodate. This yields the design principle that grounds the entire chapter: **an agentic system requires the same structural capabilities as a professional team**.

Consider how a law firm operates. Work arrives through defined channels: client calls, court filings, internal referrals. Associates must understand what partners actually want, not just what they literally said. Research requires access to the right databases with appropriate search strategies. Actions have consequences—filing a motion, sending a client letter—that require appropriate authorization. Institutional knowledge persists in case files and precedent databases. Complex matters decompose into workstreams with dependencies. Work products have completion criteria. Associates know when to escalate to partners. Teams coordinate across practice groups. And compliance controls ensure the whole operation stays within ethical and regulatory bounds.

A discretionary portfolio management team follows the same pattern. Market data and research flow through defined feeds. Analysts must interpret investment committee mandates. Research requires access to financial databases, company filings, and market intelligence. Trades have real-world consequences requiring compliance checks. Position history and investment theses persist across quarters. Portfolio construction decomposes into sector allocation, security selection, and risk management. Rebalancing has completion criteria. Analysts escalate to portfolio managers when positions approach limits. Teams coordinate across asset classes. And regulatory controls ensure fiduciary compliance.

The structural parallels are not coincidental. Both law firms and investment teams are *cognitive work systems*—organizations that process information, make decisions, and take consequential actions under uncertainty. Agentic systems are also cognitive work systems. 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.

1.2 The Ten Questions

These organizational parallels yield ten questions that any agentic system must answer—the same questions you would ask when onboarding a new professional or evaluating a team's capabilities. Table 1 maps each question to its corresponding GPA+IAT property, following the order an agent encounters them during execution.

Table 1: From GPA+IAT properties to architectural questions

Section	GPA+IAT Property	Architectural Question
Section 2	Perception	How does the agent know when it has work to do?
Section 3	Goal	How does the agent understand what is being asked?
Section 4	Perception	How does the agent find things out?
Section 5	Action	How does the agent make things happen?
Section 6	Adaptation	How does the agent remember things?
Section 7	Goal + Iteration	How does the agent break a big job into steps?
Section 8	Termination	How does the agent know when it is done?
Section 9	Termination	How does the agent know when to ask for help?
Section 10	Iteration	How does the agent work with other agents?
Section 11	All	How do we keep the agent safe?

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 start with Section 12 if you are evaluating vendor claims. We begin with the first question: how does work reach the agent?

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

Consider how work reaches a professional. A client calls with an urgent question, the court docket updates with a new filing, the calendar reminds you that a motion is due tomorrow, and a junior associate realizes an issue exceeds their expertise and brings it to your office. These four channels define how work enters your day: the phone, the inbox, the calendar, and escalation from colleagues.

Agentic systems operate in the same way. A system with tools, memory, and planning capabilities remains idle until work arrives; the architectural question is how tasks enter the system and what events trigger execution.

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 (a deadline approaching) might arrive through different channels (a calendar system, an email reminder, 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.

For governance, triggers create the audit trail. Every action an agent takes traces back to the trigger that initiated it. When a regulator asks why the system flagged a transaction, or when opposing counsel 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 audited—and in regulated practice, what cannot be audited cannot be deployed.

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).

The four channel types serve different operational needs. External feeds enable reactive monitoring, allowing the system to respond to events as they occur in the world. Human prompts enable interactive collaboration, letting the system work alongside professionals who direct its attention. Scheduled jobs enable proactive workflows that ensure routine tasks happen 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.

Before an agentic system can reason or act, it must first notice that work exists. Channels are the sensory apparatus of the system: the ways it becomes aware of its environment and the tasks it must accomplish—just as a lawyer cannot respond to a motion they never received.

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 like CM/ECF and state e-filing platforms send

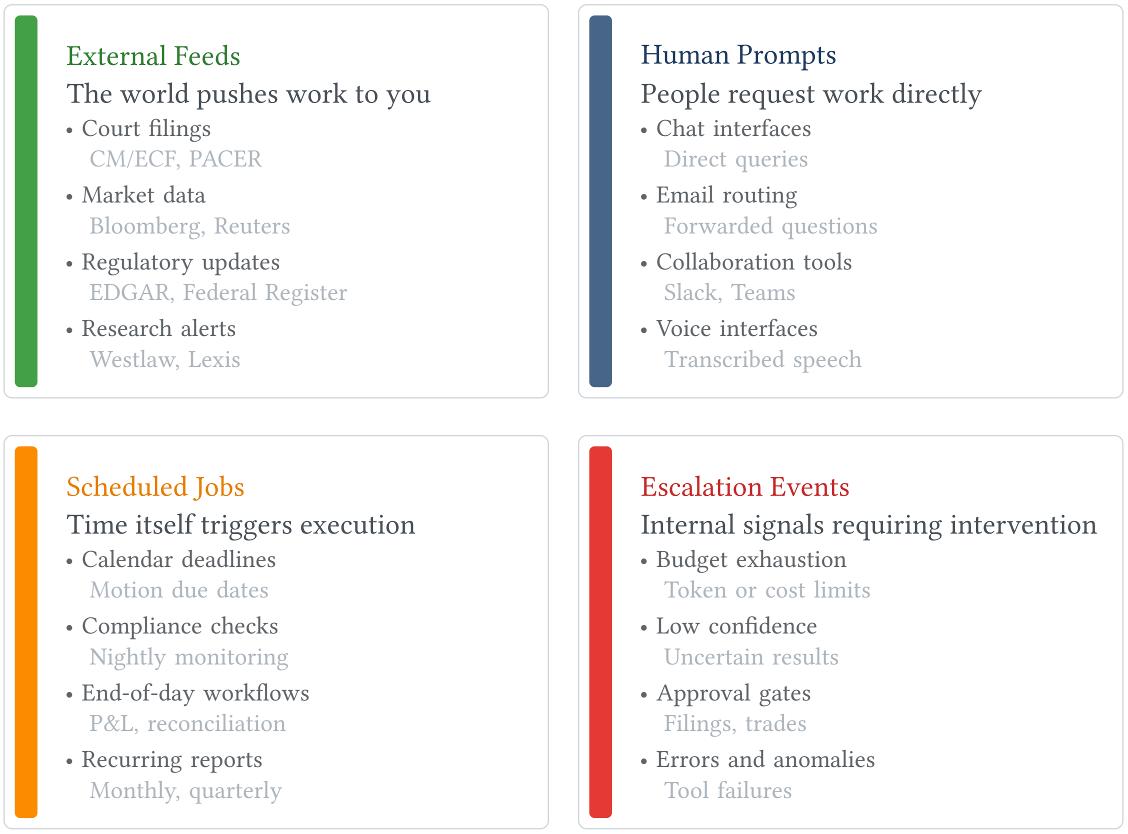


Figure 1: Four channel types through which work reaches agentic systems. External feeds push events from outside systems; human prompts arrive through interactive interfaces; scheduled jobs trigger on time-based conditions; and escalation events signal internal limits requiring human intervention. All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

notifications whenever documents are filed in cases you are monitoring. When an alert arrives, an agentic system can retrieve the filed document through PACER, analyze its contents, and trigger the appropriate response—whether that means flagging a motion for attorney review or updating a case timeline. The SEC's EDGAR system offers similar capabilities for corporate filings, allowing agentic systems to monitor competitors' 10-Ks and flag material differences from your company's disclosures. Regulatory agencies publish updates through the Federal Register and agency websites, while citator services like Westlaw and Lexis can notify the system whenever monitored cases are cited or overruled.

Financial Market Feeds: Financial institutions receive real-time market data through providers like Bloomberg and Reuters. A portfolio management agentic system can subscribe to price alerts and receive notifications when thresholds are crossed, then evaluate rebalancing rules and either execute trades within risk limits or escalate to a portfolio manager for approval. These events cascade through financial systems as trades trigger position updates, which in turn trigger risk recalculation, compliance checks, and dashboard refreshes. News feeds add another layer by delivering headlines, earnings announcements, and sentiment analytics, allowing agentic systems to assess materiality and alert managers when developments appear significant.

Speed vs. Reasoning: A Critical Distinction

Market data arrives at millisecond granularity. LLM-based reasoning operates at second-to-minute timescales. This fundamental mismatch determines where agentic systems add value in financial workflows.

Not suited for agentic systems:

- High-frequency trading and market-making
- Latency-sensitive execution
- Any task requiring microsecond response times

Well suited for agentic systems:

- Strategic portfolio decisions and rebalancing analysis
- Investment thesis development and research synthesis
- Compliance monitoring

The architecture pattern: Fast deterministic systems handle real-time data capture and threshold detection. When a threshold triggers—a position approaching its limit, a price target hit, an anomaly detected—it generates an event that the agentic system processes. Keep the microsecond path deterministic; hand off to the agentic system only once an alert is raised.

Integration Patterns: External feeds reach agentic systems through **webhooks** (HTTP callbacks for immediate notification) or **message queues** (durable event streams with delivery guarantees). Webhooks work well for low-volume, time-sensitive events where immediate delivery matters and occasional missed events are acceptable. Message queues provide ordering, durability, and replay capabilities essential for regulated applications requiring audit trails. In practice, many systems use both: a portfolio management system might use webhooks to receive immediate notification when a stock price crosses a stop-loss threshold, while using message queues to process daily trade confirmations that require guaranteed delivery and audit logging.

2.2 Human Prompts as Events

Human prompts feel different from external feeds because they are interactive and synchronous. 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 actually simplifies design, because all events can flow through the same routing and prioritization logic rather than requiring separate code paths for "chat" versus "background" work.

Chat interfaces offer the most direct channel for human interaction. An associate might type "Find Fifth Circuit authority on personal jurisdiction for e-commerce defendants," and the system searches relevant databases, presents summaries, and waits for follow-up refinements. An analyst asks for revenue growth comparisons across portfolio companies, receives a table, and requests additional filtering. What makes chat powerful is its support for iterative clarification. Each message is simply another event processed through the standard loop, just with tighter latency expectations than background tasks.

Synchronous interfaces like chat create design constraints that asynchronous channels do not. When a user is waiting for a response, every second of silence feels like something has gone wrong. Systems designed for direct interaction need to prioritize lower latency and provide transparent, regular feedback about what is happening. This might mean streaming partial responses as they are generated, displaying progress indicators that show which tools the system is invoking, or breaking complex tasks into visible steps so users can see the work unfolding. Without this feedback, a blank screen followed by a complete response thirty seconds later leaves users uncertain whether the system is working, stuck, or has crashed. Good conversational design treats feedback as a feature, not an afterthought. Acknowledge the request immediately, show progress throughout, and deliver results incrementally when possible.

Email routing lets agentic systems process work that arrives through existing communication channels. A general counsel might forward a business unit's compliance question to a monitored mailbox, and the system extracts the question, searches relevant guidance, and emails back an assessment. The main challenge here is intent classification, since email bodies tend to be unstructured and often include forwarded threads with multiple topics buried in the conversation.

Collaboration platforms like Slack and Teams allow agentic systems to appear as team members in the workflow. Users can @mention the system in channels, send direct messages, or invoke capabilities through slash commands. A litigation team discussing strategy can request research directly in their coordination channel without switching applications. Security becomes important here because collaboration platforms typically log all responses, and channels may include viewers who should not have access to certain information.

Voice interfaces work best for short, urgent requests where typing is impractical. Think of a portfolio manager checking a position while walking between meetings, or a lawyer needing a quick case citation during a call. The tradeoffs are real. Transcription errors can mangle legal jargon, turning "Chevron deference" into something unrecognizable, and authentication is harder without a keyboard. For high-stakes requests, voice interfaces should require explicit confirmation before the system takes action.

In contrast, asynchronous channels like email and background automation do not require the same real-time feedback, but they introduce different design challenges. When users are not watching, systems still need observability through logs, dashboards, and notifications that let operators understand what the system did and why. And because asynchronous requests lack the back-and-forth of conversation, getting intent right on the first pass becomes critical. A user who submits a request and walks away cannot clarify a misunderstood instruction until they see the wrong output hours later. This is why intent understanding (Section 3) and perception design (Section 4) matter even more for asynchronous workflows. The system must extract what the user actually wants from a single message, gather the right information without guidance, and produce results that match expectations without iterative correction.

2.3 Scheduled Jobs: Time as Trigger

Some work follows predictable schedules rather than arriving from external events or human prompts. End-of-day reconciliation, monthly compliance reporting, quarterly reviews, and annual filings all happen on a calendar rather than in response to some triggering event. For these recurring tasks, time itself becomes the trigger.

Calendar-driven deadlines govern much of legal practice. You must answer the complaint within 21 days, file motions 30 days before hearings, and respond to discovery within 30 days. Agentic systems can monitor litigation calendars, calculate deadlines while accounting for court holidays, schedule reminders as deadlines approach, and escalate if work remains incomplete. More sophisticated deadline systems go further by retrieving the complaint, extracting claims, generating draft answers with standard defenses, and presenting those drafts for attorney review before filing. Financial institutions face similar deadline-driven work that spans SEC reporting deadlines, tax filings, and contractual obligations to lenders.

Periodic compliance checks run even when no external event triggers a review. An investment compliance system might run nightly to check portfolios against client guidelines and flag any violations it finds. A law firm conflicts system retrieves new docket entries each day, extracts party names, and checks them against the conflicts database. These scheduled checks enable continuous monitoring that would be impractical to perform manually across thousands of matters or client accounts.

End-of-day workflows are particularly important in financial institutions, where teams need to reconcile trades, calculate valuations at market close, generate P&L reports, and prepare risk reports for the next morning. When the market closes, an EOD system retrieves final prices, marks positions to market, calculates P&L, and identifies any unexplained variances. The system then distributes reports to stakeholders, and if any step fails, it escalates rather than proceeding with incomplete data. Law firms run similar periodic workflows that remind attorneys to enter time, generate draft invoices at month-end, and flag anomalies for partner review.

2.4 Escalation Events: When Agents Reach Their Limits

The previous three channel types bring work into the agentic system from outside, but escalation events operate internally. When an agentic system reaches a limit and cannot proceed autonomously, it generates an event signaling that it requires human intervention. This transfers control to human decision-makers at precisely the moments when human judgment matters most.

Four escalation triggers appear most frequently. **Budget exhaustion** occurs when the system approaches resource limits such as token consumption, iteration counts, time limits, or cost caps, and must decide whether to stop or request additional budget. **Low confidence** triggers escalation when uncertainty is too high for autonomous action, whether due to conflicting authority, novel situations, or results that seem implausible. **Approval requirements** force escalation for certain actions that require explicit human authorization regardless of the system's confidence, such as filing court documents, sending client communications, or executing large trades. Finally, **errors and anomalies** trigger escalation when tools fail repeatedly, data is inconsistent, or the system detects red flags that require human investigation. Section 9 addresses when and how agentic systems should escalate to humans in greater detail.

2.5 Surfaces: How Users Experience Agentic Systems

A **surface** is the interaction modality through which users encounter an agentic system. The same underlying capabilities, including intent understanding, tool use, memory, and planning, can manifest through radically different user experiences. 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. But intent-based systems still require interfaces, and those interfaces shape how effectively users can express intent and consume results.

Interaction Surfaces

An **interaction surface** is the user-facing modality through which humans engage with an agentic system. The same underlying capabilities can manifest through radically different user experiences depending on four key dimensions: 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. The user types or speaks, the system responds, and the user refines based on what they see. This is the paradigm of ChatGPT, Claude, and embedded copilots. Conversational surfaces excel at *exploratory tasks* where users refine direction through iteration, such as a partner thinking through case strategy, an analyst exploring market scenarios, or an associate researching an unfamiliar area of law. The interaction is synchronous and user-initiated.

Conversational surfaces have limitations, however. They require users to articulate intent in natural language, which Nielsen calls the "articulation barrier." They lack the affordances of graphical interfaces, offering no menus to browse and no buttons to discover capabilities. And they demand attention, since the user must remain engaged throughout the interaction.

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, contract 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.

Emerging surfaces extend beyond these three patterns. *Embedded copilots* integrate agentic capabilities directly into existing applications like Word, Excel, or domain-specific software, combining familiar graphical interfaces with intent-based interaction. *Ambient interfaces* use voice or environmental sensors to enable hands-free interaction, though transcription errors and authentication challenges limit their use in high-stakes applications. *Agentic APIs* expose capabilities to other software systems rather than human users, enabling machine-to-machine orchestration.

2.6 Evaluating Trigger Systems

When evaluating agentic systems, whether you are building or buying, you should assess trigger capabilities against five criteria.

Coverage asks whether the system receives events from all relevant sources. A litigation system that monitors CM/ECF but not state court dockets has incomplete coverage that could miss critical filings. **Latency** measures how quickly events reach the system. Real-time market data requires sub-second delivery, while docket alerts can tolerate delays of several minutes. **Reliability** addresses what happens when feeds fail. Robust systems need retry logic, fallback sources, and alerting when data goes stale. **Priority mechanisms** determine whether the system can distinguish urgent events from routine ones. During a market crash or litigation crisis, the right events must reach the right handlers immediately. Finally, **auditability** ensures that every trigger is logged. When a regulator asks why the system took action, you need a complete record of the triggering event. The following chapter on governance addresses audit requirements, regulatory compliance frameworks, and governance controls in detail.

2.7 From Triggers to Action

Triggers answer how work reaches the agentic system, 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 when needed**: When should humans intervene?

The connection between questions is direct. An external feed delivers a court filing notification. The router classifies it as urgent litigation work and dispatches to the litigation agentic system. The system retrieves case context from memory, downloads the filed document through PACER, analyzes content, searches for responsive authority, generates deadline calculations, and drafts a response strategy. At each step, the system might escalate: low confidence in legal analysis triggers escalation to a senior litigator; filing a responsive document requires approval; approaching budget limits prompts a status update.

Section 3 examines the next question: once work arrives, how does the agentic system understand what's being asked?

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

When a partner walks into your office and says "look into the Johnson matter," 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; 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: natural language, often ambiguous, sometimes contradictory. The agent must extract intent: what goal is being pursued, what constraints apply, what success looks like. This is 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." "Look into the Johnson matter."

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. "Look into" 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 parsing: keyword matching, slot filling, decision trees. 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 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."

The gap between instruction and intent has always existed; what has changed is our ability to bridge it. Large language models dramatically improved intent inference from natural language. 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 desired end state that intent points toward—the outcome that would satisfy the request. Goals connect to the GPA 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: extract financial covenants, compare covenant terms to market standards, identify collateral coverage gaps, flag cross-default provisions, and draft 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.

The flow from instruction to task is: **Instruction** → **Intent** → **Goal** → **Tasks**. 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 handled work classification through explicit **routing rules**—deterministic logic that examined message attributes and dispatched to appropriate handlers. A court filing notification with `matter_id=12345` routes to the litigation system; an SEC filing by a portfolio company routes to the monitoring system. This pattern dominated middleware architecture for decades: message queues, enterprise service buses, workflow engines.

LLM-based agentic systems collapse routing into 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 has significant implications for 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 is probabilistic and may vary 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 and 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 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 certainty.

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?" "Find the latest 10-K"
- **Research and analysis**: "Research whether we can pierce the corporate veil"
- **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 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).

Structured Goal Representation: Document Review

Task type	Document review
Document	Smith Acquisition Agreement
Objective	Identify change-of-control provisions
Constraints	
Deadline	January 15, 2025
Scope	Sections 5–8
Deliverable	Summary memo
Success criteria	
•	All CoC provisions identified
•	Triggering events listed
•	Consent requirements noted

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.

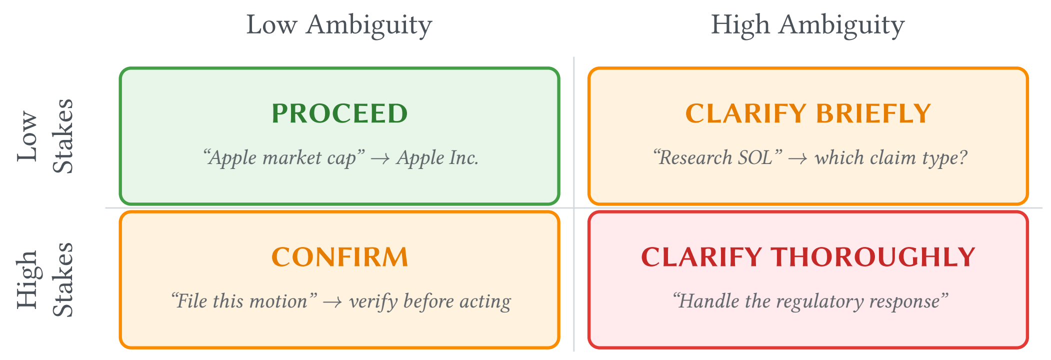

Low Ambiguity
High Ambiguity
Low Stakes
High Stakes
PROCEED
“Apple market cap” → Apple Inc.
CLARIFY BRIEFLY
“Research SOL” → which claim type?
CONFIRM
“File this motion” → verify before acting
CLARIFY THOROUGHLY
“Handle the regulatory response”

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

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.

> *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. 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.

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) help by preserving firm-specific expectations; user profiles track individual preferences; matter context provides case-specific constraints.

3.5 Validation and Domain Examples

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

Reflection and summarization involves the agent restating its understanding before proceeding, giving the user an opportunity to correct misunderstandings before work begins. Chunked validation 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?"

Extracted Goal: Credit Agreement Review

Task type	Document review
Perspective	Lender
Document	Attached credit agreement
Focus areas	Credit risk, collateral, covenants
Deliverable	Risk summary memo
Success criteria	
•	Material risks identified
•	Comparison to market terms
•	Recommendations provided

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 and long-term gains over short-term).

Extracted Goal: Portfolio Rebalancing

Task type	Portfolio rebalancing
Objective	Reduce tech exposure
Target	25%
Mechanism	Sales
Constraints	
•	Tax optimization enabled
•	Prefer loss harvesting
Approval required	Yes

The agent validates: "I'll prepare a rebalancing plan to reduce tech from 35% to 25% through sales, optimized for tax efficiency. I'll present the plan for your approval before executing any trades."

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.

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 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?

The previous section examined how agents understand what they're being asked—extracting goals from instructions, detecting ambiguity, gathering context. But understanding a task is not the same as completing it. An agent that perfectly understands "Research Ninth Circuit authority on personal jurisdiction" still needs access to case law databases. An agent that correctly interprets "Analyze this credit agreement from the lender's perspective" still needs the credit agreement.

A junior associate's effectiveness depends not just on reasoning ability but on access. The ability to query Westlaw, access Bloomberg terminals, and search the firm's precedent database determines what problems the associate can actually solve. Without access to information sources, even the most capable professional reasons in a vacuum.

Agentic systems face the same constraint. An LLM can reason about legal and financial concepts, but without *perception tools*—interfaces to external information—it cannot access current case law, market prices, client documents, or regulatory filings. Perception tools are the library card, the database subscription, and the research assistant combined: the mechanisms through which agents observe the world.

Tools and Perception

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

Perception implements the "P" in the GPA framework; it answers the question of what information the agent can access to inform its reasoning.

In this section, we examine perception: the read-only tools that enable agents to gather information. Section 5 then examines action: the write tools that enable agents to effect change. The distinction matters for governance, because read operations carry different risks than write operations.

4.1 Perception Tool Categories

Different tasks require different tools for gathering information, and effective perception depends on having the right tool for the purpose at hand. Perception tools fall into three main categories: information retrieval, document processing, and computation.

Information retrieval tools gather authoritative information from research platforms. Legal research tools include Westlaw, Lexis, Bloomberg Law, PACER for federal court filings (Administrative Office of the U.S. Courts 2024b), state court docket systems, and regulatory databases like EDGAR (U.S. Securities and Exchange Commission 2024). An agent with these tools can search case law, retrieve opinions, check citator status, and download filings. Financial research tools include Bloomberg Terminal, Reuters Eikon, FactSet, and proprietary analytics platforms. An agent with these tools can query real-time prices, retrieve fundamentals, access analyst research, and pull historical data. Internal knowledge bases round out the category, including the firm's document management system (iManage, NetDocuments), precedent databases, deal archives, and research memo repositories. An agent with access can retrieve prior work product, find template language, and check how the firm handled similar matters.

Document processing tools transform raw documents into usable information. Text extraction converts PDFs, scanned documents, and images into searchable text. OCR tools handle scanned filings; PDF parsers extract text from native documents; table extractors preserve structure from financial statements. Document classification identifies document types, which is essential in due diligence when an agent processing a data room needs to distinguish contracts from correspondence and financial statements from presentations. Entity extraction pulls parties, dates, amounts, and other structured data from unstructured documents. Extracting the borrower name, facility amount, and maturity date from a credit agreement enables structured analysis that would otherwise require manual review.

Computation tools handle perception tasks that require calculation rather than simple lookup. Deadline calculators determine response dates from rules. Federal Rules require answers within 21 days (Legal Information Institute 2024), but calculating from service date, accounting for holidays, and applying local rules requires computation rather than simple retrieval. Citation formatters convert case information into proper Bluebook format, and financial equivalents normalize identifiers (CUSIP, ISIN, ticker) across different systems. Risk metrics calculate exposure, VaR, duration, or other quantitative measures from position data. These computations inform reasoning without changing portfolios.

4.2 Model Context Protocol (MCP)

The Model Context Protocol standardizes how agents access tools (Anthropic 2024). Before standardization, every database had different commands and output formats: Westlaw worked one way, Lexis another, Bloomberg a third. MCP creates a common interface: learn the protocol once, access any compatible tool.

The architecture has three roles. The MCP Host manages the agent and controls which tools it can access, functioning like the firm's IT system determining database subscriptions. The MCP Client is the agent-side component that discovers and uses tools. The MCP Server exposes capabilities through a standardized interface; the document management system, internal knowledge base, or custom legal research tool each runs as an MCP server.

Communication follows a simple pattern: servers publish manifests declaring capabilities, clients connect through hosts, clients send structured requests, and servers return structured results.

For perception specifically, MCP defines **Resources** as read-only data access endpoints. Resources let agents:

- Query case law databases and receive structured results
- Retrieve documents from management systems
- Access market data feeds
- Search internal knowledge bases
- Fetch regulatory filings

Resources are explicitly read-only. They implement perception without enabling action. This separation enables fine-grained access control: an agent might have resource access (read documents) without tool access (file documents).

MCP Eliminates the M×N Problem

Without MCP, connecting 10 agents to 10 tools requires 100 custom integrations. With MCP, the same setup needs only 20 implementations because each agent and each tool learns the protocol once.

In legal practice, one agent queries document management systems, internal knowledge bases, and custom research tools through the same protocol. In financial services, one agent accesses portfolio systems, risk engines, and compliance databases through the same protocol.

As of late 2025, over 7,260 MCP servers have been catalogued in community directories, providing ready-made integrations for common tools.

4.3 Memory as Perception into Institutional Knowledge

Memory systems (Section 6) serve as perception tools into institutional knowledge. When an agent queries the firm's precedent database, it perceives accumulated expertise through several mechanisms.

Retrieval-Augmented Generation, or RAG, enables semantic search over document archives. The agent doesn't just keyword-match; it finds conceptually similar content. A search for "breach of fiduciary duty" retrieves documents about "violation of trust obligations" even if the exact words differ.

Vector stores power this semantic search by encoding documents as high-dimensional embeddings. The technology enables perception into large knowledge bases that would be impractical to load into context.

Prior work product becomes accessible through memory. When starting a new registration statement, the agent can perceive prior S-1 filings, SEC comment histories, and successful disclosure language. This institutional knowledge informs current work.

Memory-as-perception distinguishes experienced agents from novices. The junior associate reasons from first principles; the senior associate draws on pattern recognition from hundreds of matters. Memory gives agents access to accumulated experience.

4.4 Domain-Specific Perception Requirements

Perception for regulated professional services requires specialized enhancements across three dimensions: authority tracking, jurisdictional awareness, and confidentiality boundaries.

Authority and Provenance. Not all information is equally authoritative. Perception systems must track provenance to ensure reliable results. Authority weighting ensures that primary authority such as statutes and binding precedent ranks higher than secondary sources. When searching for "insider trading liability," a Supreme Court opinion should outrank a law review note that uses more similar language. Source verification confirms that retrieved information actually came from claimed sources rather than being hallucinated. Perception tools must return verifiable sources that can be independently checked. Currency validation ensures authority is still good law through citator integration that verifies retrieved cases haven't been overruled.

Jurisdiction and Scope. Legal and regulatory information is bounded by jurisdiction. California precedent doesn't bind Texas courts; SEC rules differ from CFTC rules. Perception must respect jurisdictional boundaries and filter appropriately. Temporal validity also matters because law changes over time. Perception systems must track effective dates, and financial temporal validity varies by context: milliseconds for trading prices, quarters for compliance effective dates. Identifier resolution handles the fact that citations appear in multiple formats ("123 F.3d 456" and "123 F3d 456" are the same case). Financial identifiers proliferate, including ticker symbols, CUSIP numbers, ISIN codes, and Legal Entity Identifiers. Perception must normalize identifiers to enable consistent retrieval.

Matter and Client Isolation. Most critically, perception must respect confidentiality boundaries. An agent working on Matter A cannot perceive documents from adverse Matter B; ethical walls must be enforced at the perception layer. In financial contexts, an agent advising Client X cannot perceive material non-public information from Client Y's engagement. Every perception must be logged, capturing what was accessed, by which agent, and for which matter. This enables compliance review and breach detection. See Section 6 for detailed treatment of isolation requirements in memory systems.

4.5 Tool Design Principles

Good perception tools follow design principles that enable reliable operation:

Single Responsibility. Each tool should do one thing well. A poorly designed tool bundles multiple functions—searching Westlaw, formatting citations, validating authority, and extracting holdings all in one interface. The untyped return value obscures what callers can expect:

Poor Design: Bundled Functions, Untyped Returns

```python
def legal_research(query: str, format: bool,
                   validate: bool, extract: bool) -> dict:
    """Returns... something. Good luck."""
    ...
```

When it fails, you can't tell which step failed. A better approach separates tools by function with typed returns, so the agent composes them and failures are isolated:

Better Design: Single Responsibility, Typed Returns

```python
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. When things go wrong—and in production, things always go wrong—tools should return informative errors. A poor approach raises generic exceptions that tell you nothing:

Poor: Opaque Exception

```python
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 legal work, graceful failure is how you avoid malpractice: when you cannot find authority, report that explicitly rather than proceeding silently.

Least Privilege and Rate Limiting. Perception tools should request only the permissions they need. A legal research tool needs read access to case databases, not write access to the document management system. If a compromised agent gains perception credentials, damage is limited to what those credentials allow. Rate limiting addresses a common failure mode: agents can get stuck in perception loops, searching repeatedly without progress. Tools should track invocation frequency and refuse requests beyond reasonable thresholds. If the agent has searched five times with no results, it should stop and escalate instead of looping indefinitely.

4.6 Evaluating Perception Capabilities

When evaluating agentic systems, assess perception against several 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. Map available perception tools against information needs to identify gaps that could limit the agent's effectiveness.

Retrieval quality measures whether the agent finds relevant information. Test with known-good queries where you know what should be retrieved, and measure both precision (relevance of results) and recall (completeness of relevant results).

Authority and provenance verification confirms that the system distinguishes authoritative from secondary sources, that you can trace retrieved information to its source, and that citations are independently verifiable.

Access controls ensure that permissions are appropriate, that the agent can access only what it should, and that confidentiality boundaries are enforced across matter and client lines. Failure handling reveals how the agent behaves when perception fails: whether it retries, tries alternatives, or escalates appropriately rather than crashing or proceeding with incomplete information.

Audit capability confirms that every perception is logged and that you can reconstruct what information the agent accessed during a task for compliance review and post-hoc analysis.

4.7 From Perception to Action

Perception enables agents to gather information, but agents must also be able to effect change: file documents, send communications, execute trades. The critical distinction is simple but important.

Perception tools are read-only. They observe without changing the world. If a perception tool fails or returns wrong results, no external state has changed; you can retry or try alternatives. Action tools, in contrast, change state. They file documents, send emails, and execute trades. Once executed, some actions cannot be undone. The risks are different, and the governance must be different as well. Section 5 examines action capabilities in detail.

5 How Does an Agent Make Things Happen?

A junior associate's job extends beyond research to producing work product. They draft memos, send emails, file documents, schedule meetings. A trader's job extends beyond analysis to execution. They enter orders, route trades, confirm allocations. The value comes 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, execute transactions. Action implements the "A" in the GPA framework.

Action Tools

Action tools enable agents to change the state of external systems. Unlike perception tools (read-only), action tools *write*: they file documents, send messages, execute trades, update databases. Once executed, some actions cannot be undone.
The distinction between perception and action is fundamental to governance. Perception risks include accessing wrong information or missing relevant data. Action risks include taking wrong actions that harm clients, violate regulations, or create liability.

5.1 Action Tool Categories

Action tools vary in consequence. The key dimension is **reversibility**: can the action be undone if something goes wrong? Four categories of action tools illustrate the spectrum from easily undone to effectively permanent.

Communication tools send information to others and are partially reversible. Internal communications include emails to colleagues, messages in collaboration platforms, and updates to internal systems. You can follow up with corrections, but you cannot unsend. External communications like emails to clients, letters to opposing counsel, and regulatory notifications carry higher stakes because recipients are outside your control. Retractions are possible but create their own problems. Automated alerts such as system-generated notifications, compliance alerts, and deadline reminders are often templated with limited customization. From a governance perspective, internal communications may proceed with post-hoc review while external communications typically require pre-approval.

Document management tools create and organize work product and are largely reversible. Document creation covers drafting memos, generating reports, and producing analysis. The documents exist internally and can be revised before distribution. Document organization includes filing in management systems, tagging and categorizing, and maintaining matter files, all generally reversible through re-organization. Template application generates documents from templates, populates forms, and produces standard documents with low risk if templates are validated. Document creation is relatively low-risk because documents can be revised before external sharing, making validation before distribution the key control.

Filing and submission tools submit documents to external authorities and are largely irreversible. Court filings through CM/ECF (Administrative Office of the U.S. Courts 2024a) or state systems become part of the public record once submitted. Amendments are possible but do not erase the original. Regulatory submissions through EDGAR (U.S. Securities and Exchange Commission 2024), FINRA, and state regulatory systems are subject to regulatory requirements, and errors can trigger enforcement. Contract execution involves signature, execution, and delivery of binding agreements that create legal obligations which may be difficult or impossible to unwind. Filing and submission require mandatory pre-approval because the irreversibility demands human verification before execution.

Transaction execution tools execute binding transactions and are effectively irreversible or costly to reverse. Trade execution involves entering orders, executing trades, and confirming allocations. Once executed, trades settle and create positions; reversal requires offsetting trades at market prices. Payment processing covers wire transfers, payment initiation, and fund disbursements. Once sent, funds are gone, and recovery requires recipient cooperation. System updates such as modifying production databases, updating client records, and changing configurations may affect downstream systems and be complex to reverse. Transaction execution requires the strictest controls: multi-factor approval, segregation of duties, and real-time monitoring.

5.2 The Reversibility Framework

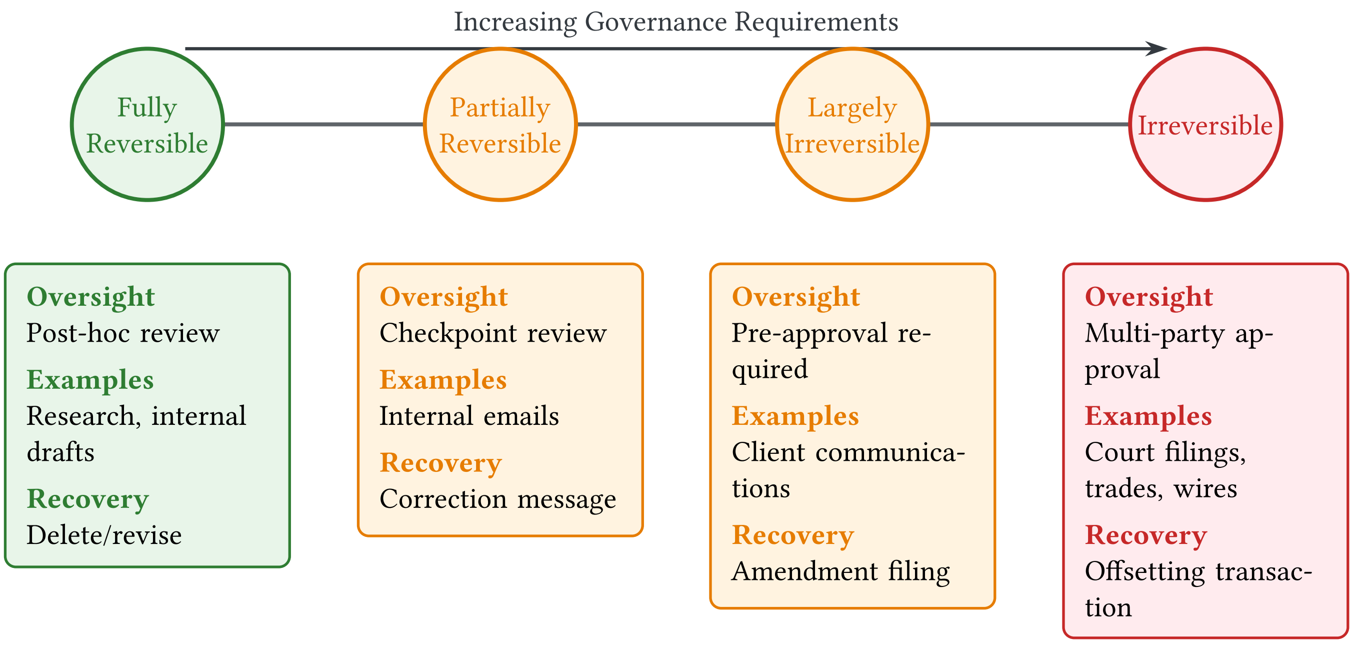


Figure 3: Action reversibility spectrum and corresponding governance requirements. As actions become less reversible, oversight shifts from post-hoc review to pre-approval and multi-party authorization. Recovery mechanisms range from simple deletion for fully reversible actions to complex offsetting transactions for irreversible ones.

Reversibility determines required oversight. Consider how you delegate to a junior associate across the reversibility spectrum.

For fully reversible actions like research and internal drafts, the associate works independently. If they make mistakes, you catch them in review. No external harm occurs. For partially reversible actions like internal communications and document organization, checkpoint review is appropriate. The associate completes work; you review before it affects others significantly. For largely irreversible actions like client communications and filings, pre-approval is required. The associate prepares; you approve before execution. For irreversible actions like trade execution and fund transfers, multi-party approval with controls is essential. Multiple people must verify before execution.

Agent governance should track reversibility. The architecture should enforce appropriate controls based on action classification.

5.3 MCP Tools and Prompts for Action

The Model Context Protocol defines two capability types relevant to action.

Table 2: Action reversibility and required oversight

Reversibility	Examples	Oversight	Recovery
Fully reversible	Research, internal drafts, calculations	Post-hoc review	Delete/revise
Partially reversible	Internal emails, document filing, alerts	Checkpoint review	Correction/follow-up
Largely irreversible	Client communications, court filings, regulatory submissions	Pre-approval required	Amendment/retraction (visible)
Irreversible	Trade execution, wire transfers, contract execution	Multi-party approval	Offsetting transaction (costly)

MCP Tools are executable functions that may change state. Unlike Resources (read-only), Tools can create and modify documents, send communications, submit filings, execute transactions, and update external systems. Tool manifests should include risk metadata: reversibility classification, approval requirements, and audit requirements. This enables hosts to enforce appropriate controls.

MCP Prompts are reusable templates for common tasks. For action tools, prompts encode standard operating procedures. Legal examples include contract review checklist prompts, due diligence workflow prompts, and filing preparation prompts. Financial examples include trade compliance check prompts, client onboarding prompts, and regulatory submission prompts. Prompts standardize action sequences, reducing variation and error. They are particularly valuable for high-stakes actions where consistency matters.

5.4 Action Security

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

All action tools must implement core security controls. Authentication verifies that the agent is who it claims to be through service accounts with strong credentials and no shared or default passwords. Authorization verifies that the agent has permission for the specific action through role-based access control and the principle of least privilege. Input validation rejects malformed or suspicious requests by validating all parameters before execution. Output confirmation requires explicit approval before high-stakes actions complete. Rate limiting caps action frequency to prevent runaway execution and escalates after repeated actions. Audit logging records every action with full context: agent identifier, action name, parameters, timestamp, result, and matter/client context.

Beyond these core controls, specific threats require targeted mitigations (OWASP Foundation 2025). Prompt injection through action parameters occurs when adversaries embed instructions in parameters the agent passes to action tools. Mitigation requires sanitizing all parameters, never passing raw user input directly to action interfaces, and validating parameter formats against strict schemas. Privilege escalation through tool chaining happens when an agent chains multiple tools to achieve capabilities no single tool grants. Mitigation requires analyzing tool combinations for escalation paths and requiring human approval for tool sequences that span security boundaries. Action replay involves capturing an action request and replaying it to re-execute the action. Mitigation requires implementing nonces or timestamps, rejecting duplicate requests, and maintaining action logs for detection.

5.5 Approval Workflows

For non-reversible actions, the agent prepares and humans approve. Several patterns implement this division of responsibility.

The single approver pattern works for routine actions with clear approval authority. The agent completes preparation and presents to one designated approver. For example: the agent prepares a court filing, presents to the supervising attorney, the attorney reviews and approves, and the agent submits.

Multi-party approval is appropriate for high-stakes actions with significant financial or legal exposure. Multiple independent approvers must sign off. For example: the agent prepares a wire transfer, operations reviews amounts and accounts, compliance reviews purpose and restrictions, the manager provides final approval, and the agent executes.

Escalating approval adjusts approval authority based on transaction size or risk. Routine actions have lower approval thresholds while exceptional actions escalate to senior personnel. For example: trades under $100K require desk manager approval, trades between $100K and $1M require senior trader approval, and trades over $1M require CIO approval.

Effective approval requests include:

- Clear description of the proposed action
- Context: why is this action needed?
- Risk assessment: what could go wrong?
- Reversibility: can this be undone?
- Supporting evidence: what analysis supports this action?
- Deadline: when is approval needed?

The approver should be able to make an informed decision from the request alone, without needing to investigate further.

5.6 Rate Limiting and Circuit Breakers

Agents can get stuck in action loops: submitting the same request repeatedly, sending multiple messages, attempting failed transactions again and again. Rate limiting and circuit breakers prevent runaway execution.

Rate limiting caps how many actions the agent can take per time period. Per-action limits might allow no more than 5 emails per minute or no more than 10 trades per hour. Per-matter limits might allow no more than 20 actions on any single matter per day without human review. Cost limits might cap transaction costs at $1,000 per session. When limits are reached, the agent pauses and escalates instead of pressing forward.

Circuit breakers automatically stop execution when anomalies are detected. If the same action fails three times, the system stops and escalates rather than retrying indefinitely. If action rate suddenly spikes, the system pauses for review since the spike may indicate agent malfunction or compromise. If cumulative actions exceed daily limits, the system stops automatically, and resumption requires human authorization. Circuit breakers transform potential runaway failures into controlled pauses that allow human intervention.

5.7 Evaluating Action Capabilities

When evaluating agentic systems, assess action capabilities against several criteria.

Start with action inventory: what actions can the agent take? Map available action tools against workflow requirements. Check reversibility classification to verify that each action is properly classified and controls are appropriate to the reversibility level.

Review approval workflows to confirm that approval gates are implemented for non-reversible actions and that approvers receive sufficient information to make informed decisions. Verify security controls including authentication, authorization, and audit logging, and confirm that penetration tests have been conducted.

Check that rate limiting is in place and matches acceptable risk tolerances. Finally, assess rollback capability: what happens when actions fail? Recovery procedures should be documented and tested.

5.8 From Action to Governance

Action tools are where agentic systems create real-world consequences. The governance implications are significant and different in kind from perception risks.

Perception risks, examined in the previous section, include accessing wrong or incomplete information. The agent reasons from bad data, but no external harm has occurred yet. Action risks include taking wrong actions that harm clients, violate regulations, or create liability. The consequences are external and may be irreversible.

This section introduced action control concepts at the architectural level: the reversibility framework, approval workflow patterns, and rate limiting mechanisms.

Within this chapter, Section 9 examines when agents should *not* act—recognizing situations that require human decision-making rather than autonomous execution. The interplay between action capability and escalation judgment is central to safe agent deployment. Beyond action, Section 6 addresses memory: how agents maintain context across sessions, learn from experience, and access institutional knowledge.

6 How Does an Agent Remember Things?

The previous two sections examined how agents gather information (perception) and effect change (action). But these capabilities operate in the moment: the agent perceives current state, reasons about it, and acts. Without memory, every interaction starts fresh. The agent cannot remember what it researched yesterday, what approaches worked or failed, what the human told it about case strategy, or what the firm has learned over decades of practice. Memory transforms an agent from a stateless tool into a system that learns and adapts.

Every experienced legal professional knows that institutional memory makes the difference between efficient work and reinventing the wheel. When you start a new securities registration matter, you do not begin from scratch. You pull the last three S-1 filings the firm completed, review the SEC comment history, and check the precedent database for disclosure language addressing similar risk factors. You do not re-research basic questions like "What are the disclosure requirements for executive compensation?" The firm maintains templates and form language that incorporate years of accumulated knowledge.

The same principle applies to portfolio management. When you revisit an equity position, you do not rebuild the investment thesis from scratch. You pull the research file, review your prior DCF model and industry analysis, then update assumptions with recent earnings data and sector trends. The accumulated research—organized and accessible—enables incremental refinement rather than duplicative work.

Memory in agentic systems serves the same purpose: context retention across sessions and learning from experience. With memory, the agent maintains continuity—much like the case file that follows a matter from initial consultation through trial—building on prior work rather than starting over each time.

Agent Memory

Agent memory stores and retrieves information across timescales. Short-term memory is the documents spread across an associate's desk during active work. Long-term memory is the firm's knowledge management system with decades of research memos. Episodic memory is the case file that tracks what happened on this specific matter. Semantic memory is the legal principles every attorney internalizes over their career.

6.1 Memory Types: From Desk to Archive

Law firms use layered filing systems, each suited to different timescales and access patterns. The associate's desk holds today's active work, the matter file contains everything related to this engagement, and the firm's precedent database archives decades of institutional knowledge. Each layer trades immediacy for capacity, and effective practice requires knowing which system to consult when. Agent memory systems follow the same layered pattern.

Working memory in a law firm looks like the papers spread across an associate's desk: the documents actively in use right now. In agentic systems, working memory takes the form of the *context window*, the tokens currently loaded in the LLM's attention. Just like desk space, context windows have strict limits. The associate can only have so many documents open at once; the agent can only hold so many tokens in active context (as of late 2025, 200K tokens for leading models, though this ceiling continues to rise). When the case involves more documents than fit on the desk, you need other storage systems. The banker's equivalent is market data on the trading screen: live prices, recent news, positions from today's session. This too is working memory: fresh and immediately accessible but gone when the session ends.

Episodic memory corresponds to the matter file for the specific engagement. Every memo, every piece of correspondence, every research result related to this matter goes in the file. The associate does not re-research questions already answered; the file comes first. When the partner asks "What's our argument on venue?," the associate pulls the file and reads the prior research memo rather than starting over. In agentic systems, episodic memory captures the history of actions and outcomes for a specific task or session. The agent remembers: "I searched for Ninth Circuit venue cases, found three relevant opinions, drafted analysis, partner reviewed and approved." When asked a follow-up question, the agent retrieves that prior work. The financial parallel is the research file for each position. When you revisit a stock you analyzed six months ago, you do not rebuild the entire investment thesis. You pull the file, read your prior analysis, and update it with new information. The agent does the same: retrieve prior analysis, check what has changed, update conclusions.

The third layer is the firm's precedent database: institutional knowledge accumulated over decades. Every time the firm handles a particular type of matter, the work product goes into the archive. When you need language for a force majeure clause in a construction contract, the precedent database offers fifty examples from prior deals. When you need briefing on qualified immunity, the database contains the firm's best arguments from the past ten years, organized by circuit and issue. Agentic systems implement this through **retrieval-augmented generation (RAG)**: dynamically fetching relevant information from a large corpus to augment the agent's reasoning (Lewis et al. 2020). RAG enables agents to access institutional knowledge beyond what fits in context. For the financial analyst, the equivalent is the firm's market research database: historical earnings reports, industry analyses, competitive landscape studies, and valuation models, all searchable and retrievable when analyzing new opportunities.

The technology that powers RAG is the **vector store**, which makes precedent databases searchable semantically rather than just by keyword. Rather than matching exact terms, which misses synonyms and related concepts, vector stores encode documents as high-dimensional embeddings that capture semantic meaning (Reimers and Gurevych 2019). When you search for "breach of fiduciary duty," the system finds not just documents containing that exact phrase but also documents about "violation of trust obligations" or "failure to act in good faith," concepts that mean similar things even if worded differently. Each memory layer trades speed for capacity: working memory is fastest but smallest, vector stores are largest but require retrieval latency.

6.2 Retrieval-Augmented Generation

RAG enables agents to access institutional knowledge, the equivalent of asking the firm librarian "show me our best research on this issue." Traditional keyword search works but misses cases discussing the same concept using different language. Semantic search using embeddings finds conceptually similar content even when exact words differ.

The RAG pipeline has four steps, each transforming information to enable semantic retrieval. First, chunking breaks documents into semantic units while preserving metadata, so a 50-page contract becomes many retrievable segments, each maintaining reference to its source location and document context. Second, embedding converts each chunk into a high-dimensional vector that encodes semantic meaning, positioning similar concepts near each other in vector space regardless of the specific words used. Third, retrieval finds chunks similar to the query by embedding the user's question and returning chunks with similar vectors. A question about "breach of fiduciary duty" retrieves chunks discussing "violation of trust obligations" even though the exact phrase never appears. Finally, generation augments the agent's prompt with retrieved content, so the LLM sees both the question and relevant context from the knowledge base when formulating its response.

The best implementations enhance basic RAG with advanced patterns. Hybrid retrieval combines semantic search (embeddings) with keyword search (BM25, a standard term-frequency ranking algorithm (Robertson and Zaragoza 2009)), catching both conceptual similarity and exact term matches that pure semantic search might miss. Query rewriting expands ambiguous queries before retrieval, transforming vague questions like "What's the rule?" into specific queries such as "What is the legal rule governing [topic from context]?" based on conversational context. Reranking scores results by authority after initial retrieval, ensuring that binding precedent ranks above secondary sources even when secondary sources use more semantically similar language. Filtered retrieval constrains results by jurisdiction, time period, or other metadata, preventing the system from retrieving California cases when researching New York law or outdated regulations when current guidance is required.

One requirement is critical: never let fabricated citations reach the user. Hallucinated citations, plausible-sounding but nonexistent cases, are a known failure mode—studies have found that leading AI legal research tools hallucinate 17–33% of the time even with RAG (Dahl et al. 2024). Before any citation reaches work product, verify that the source actually appeared in retrieved context.

6.3 Domain-Specific Memory Considerations

Memory for regulated professional services requires specialized enhancements beyond generic RAG implementations. The systems must account for authority hierarchies, jurisdictional boundaries, temporal validity, and identifier normalization, requirements rarely found in consumer applications but critical for professional practice.

Authority weighting ensures that not all information is treated equally. Primary authority (statutes, binding precedent) should rank higher than secondary sources. When searching for "insider trading liability," a Supreme Court opinion should outrank a law review note using more similar language. Financial systems apply similar authority weighting: SEC no-action letters rank above law firm client alerts, official exchange rules above broker-dealer summaries, and Federal Reserve guidance above market commentary.

Jurisdiction awareness respects legal boundaries. California precedent doesn't bind Texas courts; SEC rules differ from CFTC rules. Metadata tagging during ingestion enables proper filtering. An agent researching Delaware corporate law must not surface New York case law as controlling authority, even if semantically similar.

Temporal validity matters because law changes. Citator integration validates that retrieved cases haven't been overruled. A 1985 securities case may have been good law for decades but reversed in 2023; RAG must surface the current state of the law, not historical precedent. Financial temporal validity varies by context: milliseconds for trading data (stale prices cause losses), days for research reports (quarterly updates suffice), and effective dates for compliance rules ("What are the margin requirements *as of January 15, 2025*?"). When does memory become stale? For legal research, staleness depends on dynamism: tax law changes annually, constitutional doctrine evolves slowly. For financial data, equity prices are stale in seconds, but industry structure analysis remains valid for quarters. Effective memory systems tag temporal validity and trigger refresh when content ages beyond acceptable bounds.

Identifier resolution normalizes citations ("123 F.3d 456" and "123 F3d 456" are the same case) and financial identifiers. Tickers change over time, and companies have multiple identifiers: CUSIP (Committee on Uniform Securities Identification Procedures numbers), ISIN (International Securities Identification Numbers), and LEI (Legal Entity Identifiers). Without normalization, retrieval fragments: half your precedent on *Smith v. Jones* does not surface because some associates cited it with spacing variations.

6.4 Matter and Client Isolation

Most critically, matter and client isolation prevents memory from one matter leaking into another. Law firms maintain ethical walls between conflicted representations; if an agent uses Matter A's privileged information on adverse Matter B, that's a privilege waiver and potential malpractice. Financial isolation prevents material non-public information (MNPI) exposure. An agent advising on Company X's acquisition cannot access research files containing MNPI about Company X from a separate advisory engagement.

Implementing separation at the memory layer requires four controls. Separate namespaces give each matter its own isolated memory partition. Retrieval queries are scoped to the current matter's namespace, preventing cross-matter leakage. Access controls use role-based permissions to determine which humans and agents can access which memory namespaces. Only team members assigned to Matter A can read Matter A's episodic memory or RAG results. Audit trails log every memory read and write with timestamp, user/agent identity, matter identifier, and data accessed. This enables post-hoc compliance review and breach detection. Secure deletion ensures that when a matter closes or a client relationship terminates, memory is permanently and verifiably deleted, not just logically marked inactive. Financial regulations often mandate retention schedules but also mandate destruction after expiration.

For legal contexts, matter isolation maps to ethical walls. For financial contexts, information barriers prevent trading on MNPI or front-running client orders. Both domains treat memory isolation as a *regulatory compliance requirement*, not merely an engineering best practice; Section 11 previews the governance frameworks.

6.5 Evaluating Memory Systems

How do you know if memory works? Testing should cover retrieval quality, isolation integrity, temporal validity, and performance under scale.

Retrieval quality measures precision (are retrieved documents relevant?) and recall (did retrieval find all relevant documents?). For legal research, gold-standard test sets come from known-good research memos: given the research question, does RAG retrieve the same primary authorities the human researcher cited? For financial analysis, compare retrieved earnings reports and industry studies to what an analyst would manually pull.

Isolation integrity verifies that cross-matter queries return zero results. Matter A's agent should never retrieve Matter B's documents, even if semantically similar. Audit logs should show no unauthorized access attempts. Red-team testing deliberately attempts privilege breaches to validate controls.

Temporal validity tracking measures retrieval freshness. How often does the system surface overruled precedent or stale financial data? Measure lag between legal/regulatory change and knowledge base update. For high-stakes domains, daily or real-time refresh may be required.

Performance under scale matters as episodic memory grows. A multi-year litigation generates thousands of documents. Does retrieval latency degrade? Can the system handle concurrent queries across hundreds of active matters without contention?

6.6 From Memory to Planning

Memory provides the context agents need to plan effectively. Without memory, agents repeat failed strategies because they cannot recall what did not work. Research starts from scratch even when prior work exists. Preferences and constraints must be re-specified each session, and institutional knowledge remains inaccessible.

With memory, agents learn from experience. The agent recalls: "Last time I searched with broad terms, I got 10,000 results. This time I'll use narrower queries." Prior work product is retrieved and built upon rather than duplicated. User preferences persist across sessions, and firm expertise informs every task.

Memory enables adaptation—the "A" in the GPA+IAT framework. The agent's behavior improves over time precisely because it learns from experience.

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 and execution monitoring.

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

A litigation partner approaching a new matter does not start by drafting motions. The partner develops a strategy: discovery first (what facts do we need?), then dispositive motions if the law clearly favors us, settlement discussions in parallel, trial prep as a backstop. Discovery breaks into phases: initial disclosures, document requests, interrogatories, depositions. Tasks distribute across the team: senior associate handles briefing, junior associate does document review, paralegal manages scheduling and filings. Throughout, the partner monitors progress: are we on track for deadlines? Are discovery responses revealing helpful facts or should we adjust our theory?

This is **planning**: decomposing complex goals into action sequences, much like 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 know when they're done.

Planning

Planning decomposes complex goals into sequences of actions. It encompasses:

- **Decomposition**: Breaking large tasks into manageable steps
- **Sequencing**: Ordering steps logically (what depends on what?)
- **Allocation**: Assigning steps to tools or agents
- **Monitoring**: Tracking progress toward the goal
- **Adaptation**: Adjusting the plan when circumstances change

Without planning, an agent is like an associate who keeps running searches without a research 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). The partner asks for authority that a forum selection clause is unenforceable. The associate reasons: "Key grounds are unconscionability and public policy. Start with *Atlantic Marine*." They search, observe results, reason again: "The unconscionability cases involve consumer adhesion contracts—not our commercial situation. The public policy line is closer." They search again, refine based on results.

Each cycle has three components:

- **Thought**: Explicit reasoning about what to do next
- **Action**: Tool call to gather information or effect change
- **Observation**: 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—research questions, fact investigation, market analysis.

Plan-Execute. This pattern separates planning from execution. For document review ("Review 50 contracts for choice-of-law, forum selection, arbitration, and liquidated damages provisions"), the associate makes a plan: checklist of provisions, open each contract, record findings. Then they execute systematically. The plan does not change because the task is well-defined.

Plan-Execute fits workflows with established procedures: due diligence checklists, compliance reviews, document assembly. You create the plan upfront and execute methodically. Research variants like ReWOO (Xu et al. 2023), which separates reasoning from observation to reduce token usage, and LLMCompiler (Kim et al. 2024), which optimizes execution graphs for parallelism, enable parallel tool calling when steps are independent, though the basic pattern remains: plan first, then execute.

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

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 one agent)
- Prepare exhibits (another agent)
- Draft jury instructions (another agent)

Each specialist may decompose further. This enables parallelization and specialization, mirroring how litigation teams work with multiple associates and paralegals handling different workstreams simultaneously. Section 10 provides detailed treatment of multi-agent coordination patterns.

The planning patterns above represent a fundamental shift from traditional workflow automation. Traditional workflow engines used *static orchestration*: predefined graphs specifying which steps follow which, what conditions trigger branches, and how exceptions route to handlers. Business process management (BPM) systems, enterprise service buses, and workflow automation tools all embodied this pattern. The workflow was designed at build time; at runtime, the engine merely executed it.

Static vs. Dynamic Orchestration

Static orchestration executes predefined workflow graphs. The same input always produces the same execution path. Predictable, auditable, but inflexible.

Dynamic orchestration reasons about task decomposition at runtime. The LLM examines the goal, considers available resources, and constructs an appropriate plan on the fly. Adaptive, but less predictable.

LLM-based orchestration is inherently dynamic. "Prepare for trial" might decompose differently depending on case complexity, available resources, and timeline. The LLM examines the goal, considers available specialist agents, and constructs an appropriate delegation structure on the fly. This adaptability is both the promise and the challenge of agentic planning.

The distinction has practical implications. Static workflows handle anticipated scenarios well, but novel situations fall through to error handlers or human intervention. Dynamic orchestration adapts to the specific task, potentially handling situations the system designer never anticipated. Maintenance costs also differ: static workflows require explicit updates when processes change, while dynamic orchestration absorbs changes through prompt updates or model fine-tuning.

Auditability Challenge

Static workflows produce predictable execution traces; given the same input, the same workflow executes. Dynamic orchestration may produce different decompositions for similar inputs, complicating audit and compliance. For regulated applications, log the *reasoning* behind orchestration decisions, not just the decisions themselves.

Production systems often combine both approaches. High-volume, well-understood processes use static workflows for efficiency and predictability. Complex, variable, or novel tasks use dynamic orchestration. The static layer handles routine work; the dynamic layer handles everything else. The planning patterns described above—ReAct, Plan-Execute, Hierarchical—are all forms of dynamic orchestration: the LLM decides what to do based on the task at hand, rather than following a predetermined script.

7.2 Choosing the Right Planning Pattern

Selecting the right pattern depends on task structure and required autonomy level:

Table 3: 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 work-streams	Hierarchical	Distributed	M&A transaction, portfolio construction, multi-jurisdiction filing

The autonomy column matters for governance. Higher-autonomy patterns require more sophisticated oversight.

Plan-Execute patterns operate with moderate autonomy. The agent works within tight bounds defined by the plan. Oversight focuses on plan validation and output review. ReAct patterns involve higher autonomy because the agent makes decisions about what to search, what to pursue, and when to stop. Oversight requires explicit termination mechanisms, confidence thresholds, and reasoning trace review. Hierarchical patterns distribute autonomy across multiple agents making decisions. Oversight requires clear delegation contracts, escalation paths between agents, and coordination monitoring. The principle is simple: match oversight rigor to autonomy level.

7.3 Understanding the Task Before Planning

Before planning, agents must understand what they're being asked to do. Section 3 covers intent extraction in detail. For planning purposes, the key outputs are task classification, constraints, and success criteria.

Task classification determines which planning pattern to use: is this exploratory (ReAct), structured (Plan-Execute), or complex (Hierarchical)? Constraints define what bounds the work: deadlines, budgets, and scope limitations. Success criteria answer the question of how the agent will know when it's done and what deliverable is expected.

Effective planning requires clear inputs. Ambiguous goals produce unfocused plans; unclear success criteria make termination difficult.

7.4 Budget Architecture

Without explicit resource budgets, agents can run indefinitely. This is the "runaway associate" problem: you asked for two cases, the associate gives you fifty because they did not know when the answer was sufficient. This subsection examines budget allocation as a planning mechanism; Section 8 examines what happens when budgets are exhausted and how agents recognize resource limits and terminate gracefully.

Four budget types provide control over agent execution, each addressing a different dimension of resource consumption. Token budgets limit LLM API consumption, preventing expensive runaway reasoning loops where the agent keeps elaborating without making progress. Time budgets enforce deadlines by stopping execution after a fixed duration, perhaps 10 minutes, if no meaningful progress has occurred. Tool call budgets prevent runaway tool loops by capping the number of external calls; after 20 searches without progress, the agent should escalate rather than continuing to search. Cost budgets cap total spending in dollars, particularly important when using expensive models or external APIs where unconstrained execution could generate substantial charges.

These budgets cascade through levels: session budgets constrain entire engagements, task budgets allocate resources to specific work items, and subtask budgets subdivide further. A legal research task might receive a 30-minute time budget and 50,000-token limit; if it spawns subtasks, those subtasks share the parent budget rather than each receiving unlimited resources.

Token costs compound across agentic workflows. Consider a credit facility review: a 200-page document requires roughly 80,000 tokens to ingest. Each section analysis might consume 10,000–20,000 tokens across reasoning and tool calls. Retrieval from precedent databases adds tokens. Multi-iteration refinement multiplies costs. A comprehensive review might consume 500,000–1,000,000 tokens. At illustrative pricing (late 2025: roughly \$3–15 per million input tokens for leading models; verify current rates), that's \$2–15 per review in API costs alone, before infrastructure, storage, or human review time. For portfolio management running continuously, costs accumulate differently:

thousands of small queries per day rather than occasional large tasks. Monitor aggregate daily and weekly costs, not just per-task.

The economics of agent assistance depend on task type. Retrieval-heavy tasks like research and document review show the clearest ROI when agents reduce hours substantially. Judgment-intensive tasks show less clear ROI when extensive human revision is required. The critical variable is human review time: agent output requiring extensive correction may cost more than human-only work. Billing norms are evolving: some firms pass efficiency gains through as reduced hours, others add technology fees, others use fixed-fee arrangements. ABA Formal Opinion 512 requires competence regardless of tools and reasonable billing (American Bar Association Standing Committee on Ethics and Professional Responsibility 2024). Transparency about AI assistance enables clients to evaluate the value proposition.

When budgets tighten, agents should degrade gracefully rather than failing completely. Tiered outputs provide value at every budget level: minimal budget delivers the controlling statute with citation; moderate budget adds key holdings; full budget delivers comprehensive analysis. The user receives something useful regardless of where termination occurs. Soft limits at 75–80% of budget warn the agent that resources are running low, prompting it to prioritize completion over exploration. If the agent has found adequate authority, it should synthesize rather than searching for more. Hard limits at 100% terminate execution and return whatever partial results exist. A budget-aware agent that delivers partial results is more useful than one that fails completely, and partial results often suffice for the user's immediate needs.

7.5 Knowing When to Stop

Perhaps the most critical planning capability is knowing when to stop. Section 8 covers termination in detail; for planning purposes, four categories of stopping conditions guide agent behavior.

Success conditions apply when the goal is achieved: the research question is answered, the document is reviewed, the analysis is complete. The agent returns its result and stops. Resource exhaustion occurs when budget limits are reached; the agent returns partial results or escalates for additional allocation. Confidence thresholds trigger escalation when uncertainty is too high for autonomous action, routing the task to human review rather than proceeding with unreliable conclusions. Error conditions indicate that repeated failures suggest a problem that retrying will not solve.

Define explicit stopping rules, just as you would instruct an associate: "If you find three on-point circuit opinions that all agree, you're done. If you've searched for two hours and found nothing, come talk to me." Agents need the same clarity about when their work is complete.

7.6 Guardrails and Loop Detection

Even with budgets and termination conditions, agents can get stuck in unproductive loops. Multiple mechanisms detect and prevent these patterns: step limits, reflection checkpoints, external watchdogs, and meta-policies. Section 8.4 examines loop detection and guardrail mechanisms in the context of termination.

7.7 From Planning to Termination

Planning answers how agents decompose work, but every plan must end. The next two questions address the boundaries that contain autonomous execution.

Termination (Q7, Section 8) answers the question of how an agent knows when it is done. This requires defining success criteria so the agent can recognize completion, establishing budget limits so the agent cannot run indefinitely, and implementing completion recognition so the agent delivers results rather than continuing to refine. Escalation (Q8, Section 9) answers the question of how an agent knows when to ask for help. This requires confidence thresholds that trigger human review when uncertainty is high, authority boundaries that prevent agents from exceeding their mandate, and human-in-the-loop integration that makes escalation smooth rather than disruptive.

Without clear termination, agents run forever. Without escalation, agents exceed authority. These boundaries define the safe operating envelope for autonomous execution, ensuring that agents remain useful tools rather than becoming uncontrolled processes.

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

Every professional learns to recognize completion. The research memo is done when you've found sufficient authority and synthesized it coherently. The due diligence is done when you've reviewed all material documents and reported findings. The trade is done when the order executes and settles. Knowing when work is complete—and when it isn't—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 asked for two relevant cases, the associate gives you fifty because they did not know when enough was enough.

Termination

Termination conditions define when an agent should stop executing. Three outcomes are possible. Success means the goal is achieved and the agent delivers the result. Failure means the goal cannot be achieved, so the agent reports why and stops. Escalation means the agent cannot determine success or failure, transferring the decision to human judgment.

Termination implements the "T" in the GPA+IAT framework. Without termination, agents lack the sixth property that distinguishes agentic systems from runaway processes.

8.1 Termination Condition Categories

Five categories of termination conditions bound agent execution: success conditions, resource budgets, confidence thresholds, error conditions, and escalation triggers.

The most obvious termination is when the goal is achieved and the agent can return the result. Success conditions take several forms. Completeness criteria ask whether all required elements have been produced. For document review, this means all provisions on the checklist have been analyzed. For research, the legal question has been answered with supporting authority. For portfolio rebalancing, allocations match targets within tolerance. Quality thresholds ask whether the output is good enough. For a research memo, are conclusions supported by binding authority? For a risk assessment, have material risks been identified and analyzed? Quality thresholds often require human judgment because the agent can check completeness but may not assess quality reliably. Convergence criteria ask whether the agent has stopped learning new information. If the last three searches returned no new relevant authority, the research may be saturated. If the last five portfolio adjustments produced diminishing improvement, optimization may have converged.

Resource budgets provide hard limits that prevent runaway execution by capping consumption across multiple dimensions. Section 7.4 examines budget architecture as a planning mechanism; here we focus on budgets as termination triggers. Token budgets stop execution after a specified threshold, say 50,000 tokens, preventing expensive reasoning loops that would otherwise continue indefinitely. Time budgets enforce deadlines by terminating after a fixed duration, ensuring that research tasks do not consume an entire day when an hour was expected. Iteration budgets cap tool calls, stopping after twenty searches to prevent infinite loops where the agent keeps trying slightly different queries without progress. Cost budgets provide the most direct control, halting execution after spending a dollar amount (perhaps $5 in API calls) to limit financial exposure. These budgets cascade and interact: a task might hit its time limit before exhausting its token budget, or vice versa. Budget exhaustion does not mean failure; partial results may still be valuable. But the agent must stop and report instead of running indefinitely.

Confidence thresholds gate actions on certainty, creating a decision boundary between autonomous execution and human review. When confidence is high, the agent delivers its answer. When confidence drops below a calibrated threshold, perhaps 80%, the agent stops and escalates rather than proceeding with uncertain information. This mirrors how associates should work: "I'm not confident this is right. Let me ask the partner before proceeding." Calibrating these thresholds is challenging. Research suggests that language models can often predict their own accuracy but require careful prompting (Kadavath et al. 2022). Agents may be overconfident, proceeding when they should escalate, or underconfident, escalating unnecessarily and providing no value. Effective calibration requires testing against known outcomes, comparing agent confidence to actual accuracy, and adjusting thresholds until the agent escalates at appropriate uncertainty levels.

Error conditions require agents to recognize when things are going wrong and terminate rather than compounding errors. Repeated tool failures signal infrastructure problems: if Westlaw times out three times consecutively, the agent should stop rather than retrying indefinitely while consuming budget. Inconsistent data, such as a revenue figure in the 10-K that does not match the earnings release, requires human investigation, not agent guesswork about which source is correct. Constraint violations demand immediate termination: if the planned action would exceed position limits or breach confidentiality, the agent must stop before acting, not after. Some tasks prove impossible as specified. Analysis may reveal conflicting requirements, missing prerequisites, or logical impossibilities. In these cases, the agent should report the impossibility honestly rather than proceeding with a compromised approach that satisfies the letter of the instruction while violating its spirit.

Escalation triggers require human judgment regardless of whether the task has succeeded, failed, or consumed its budget. Novel situations that do not match training patterns need human expertise to navigate. High-stakes decisions warrant human approval even when the agent is confident, because the consequences of error justify the overhead of review. Authority boundaries define what the agent can do autonomously; actions beyond those boundaries require escalation by design, not because something went wrong. Section 9 examines escalation patterns in detail.

8.2 Defining Success Criteria

Vague goals produce unclear termination. "Research the statute of limitations" could mean finding one relevant case or exhaustively surveying all circuits. Effective success criteria take several forms, each providing a different signal that work is complete.

Completeness checklists enumerate what must be delivered. For credit agreement review, the checklist might require identifying all financial covenants, comparing them to market terms, flagging provisions that differ from the firm's template, and summarizing material risks. The agent terminates when all checklist items are complete. Sufficiency thresholds define "enough" without requiring exhaustive coverage. For case research, sufficiency might mean finding at least three on-point circuit opinions, or if circuits conflict, identifying the leading case from each side. The agent knows when sufficiency is reached without searching every database.

Convergence criteria recognize diminishing returns. If three consecutive searches return no new relevant authority, the research may be saturated; further searching is unlikely to yield value. Deliverable specifications define the output format—a two-page memo with executive summary, analysis, and recommendation—so the agent knows what success looks like, not just what success requires.

Consider how experienced attorneys instruct associates: "If you find clear Ninth Circuit authority, you're done. If the circuits are split, map the split and recommend which approach applies to our facts. If you can't find binding authority after two hours, come talk to me." Agents need the same clarity.

8.3 Recognizing Failure

Not every task succeeds, and agents must recognize failure and report it honestly. Negative results are still results: "I searched all major databases and found no authority on point" is a valid finding that the attorney needs. The absence of authority is information. Agents should report negative results explicitly instead of searching indefinitely in hope of finding something.

When failure occurs, diagnostic reporting explains what was attempted and why it failed. A useful failure report might state: "I searched Westlaw, Lexis, and Bloomberg Law using [specific queries]. Zero results suggest either the issue is novel or the search terms are wrong. Recommend manual review of secondary sources or consultation with practice group expert." This gives the human actionable information rather than a bare "task failed" message.

Partial completion should be acknowledged honestly. If the agent completed analysis of Articles 1-4 before a tool failure, it should report what was accomplished and what remains: "Articles 5-8 remain unanalyzed." Partial results may still be valuable, and preserving them prevents wasted effort. Where possible, root cause identification helps humans decide next steps: Was this a tool failure that will resolve itself? Impossible requirements that need rethinking? Insufficient information that requires additional input? The agent's diagnosis informs the human's response.

8.4 Guardrails and Loop Detection

Even with well-defined termination conditions, agents can get stuck in unproductive loops—searching repeatedly without progress, rephrasing queries slightly, finding nothing, rephrasing again. Multiple mechanisms detect and prevent these loops.

Step limits provide the simplest guardrail: after N steps, stop and require human approval to continue. This prevents unbounded execution regardless of what the agent thinks it is accomplishing. Progress detection monitors whether recent actions produced value: if the last five actions yielded no new information, the agent may be stuck and should trigger reflection or escalation. Reflection steps build self-assessment into the workflow, periodically asking meta-questions: "Am I making progress toward the goal? Have my recent actions been productive? Should I try a different approach or escalate?"

External watchdogs monitor agent behavior from outside the agent's own reasoning. If the same tool is called repeatedly with nearly identical parameters, an external system can recognize the loop pattern and intervene. Meta-policies encode loop detection rules directly: calling the same tool with the same parameters more than three times is probably a loop, so stop and escalate.

Without loop detection, agents will eventually get stuck in production. The question is not whether it will happen, but whether you will detect it when it does.

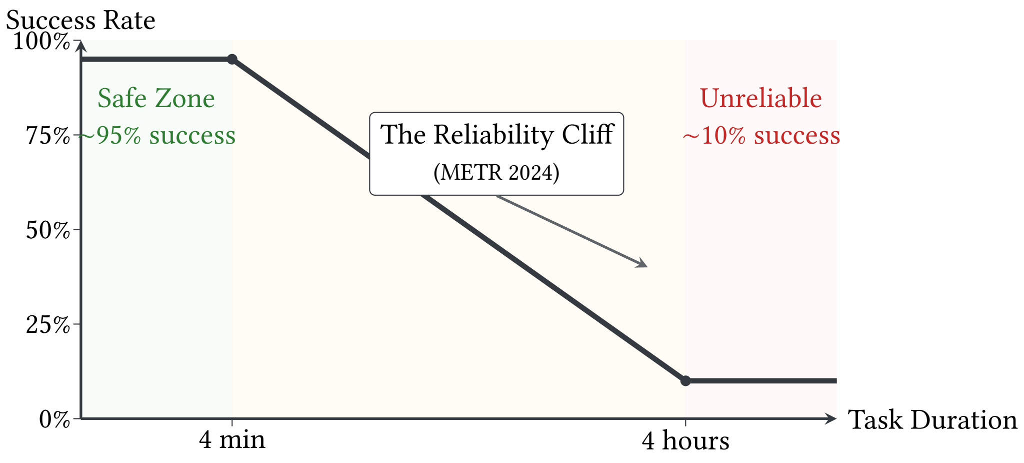
8.5 The Reliability Cliff

Independent benchmarking reveals a sharp reliability boundary. METR (Model Evaluation and Threat Research) tested agents across standardized task suites varying in duration and complexity. The results:

The Four-Minute Cliff

METR's 2025 study found that agents achieve **near-perfect success on tasks under 4 minutes**, but **under 10% success on tasks over 4 hours** (METR 2025).

This gap, from near-100% to under-10%, defines the current boundary between reliable and unreliable agent deployment. The implication is clear: decompose tasks aggressively, keep individual agent tasks short, insert human checkpoints between phases, and do not expect autonomous completion of multi-hour workflows.



Implication: Decompose tasks to stay under 4 minutes for reliable execution

Figure 4: The Reliability Cliff: Agent success rates drop dramatically as task duration increases. METR (2024) found that agents maintain approximately 95% success on tasks under 4 minutes, but success rates fall to roughly 10% for tasks exceeding 4 hours. This sharp transition motivates the decomposition of complex tasks into shorter subtasks and the use of aggressive timeout policies in agent architectures.

The reliability cliff has several causes, each contributing to the dramatic drop in success rates as task duration increases. Compounding errors are perhaps the most fundamental: each step introduces error probability, and these probabilities multiply. A 95%-accurate retrieval step followed by 90%-accurate reasoning followed by 85%-accurate action yields roughly 73% end-to-end accuracy, before accounting for sequencing decisions. Over a four-hour task with dozens of steps, these compounding errors accumulate into near-certain failure.

Planning fragility compounds the problem. Agents frequently select suboptimal tool sequences, get stuck in loops, or fail to recognize when their approach is not working. A human would step back and reconsider; agents often persist with failing strategies. Integration brittleness adds another failure mode: tool APIs return unexpected formats, authentication tokens expire, rate limits trigger. Each integration point is a potential failure mode, and complex tasks touch many integration points.

Design for this reality. Decompose aggressively. Validate at checkpoints. Assume agents will fail and design for graceful degradation.

8.6 Graceful Degradation

When termination occurs before full completion, agents should degrade gracefully rather than failing completely. Tiered outputs provide value at every budget level: at 20% budget, deliver the controlling statute with citation; at 60% budget, add key holdings; at 100% budget, deliver comprehensive analysis. Partial results are better than nothing, and users can decide whether partial results suffice or warrant additional investment.

Progress preservation saves intermediate state so humans can resume where the agent stopped. If the agent analyzed 30 of 50 contracts before budget exhaustion, that work should not be lost when execution terminates. Clear status reporting communicates exactly where things stand: "Completed 60% of task. Remaining: Articles 5-8 unreviewed due to budget exhaustion. Findings so far: [summary]." The human knows what was accomplished and what remains.

Beyond reporting status, agents should recommend next steps when possible. "Recommend allocating additional 30 minutes to complete review" gives the human a concrete decision to make. "Remaining work is routine; recommend proceeding with partial findings" helps humans assess whether additional effort is worthwhile. The goal is a handoff that enables informed human decision-making, not a handoff that forces the human to start over.

8.7 Evaluating Termination Capabilities

When evaluating agentic systems, assess termination capabilities against six criteria that distinguish robust systems from fragile ones.

Start with success criteria clarity. Are termination conditions explicit? Can you predict when the agent will stop? Systems with vague or implicit termination conditions produce unpredictable behavior. Test budget enforcement by setting tight budgets and verifying the agent actually stops; some systems log budget exhaustion but continue anyway. Assess loop detection by presenting impossible tasks or unavailable tools; a system without loop detection will spin indefinitely.

Examine failure reporting to see whether failures are actionable. When tasks fail, does the agent explain why? A bare "task failed" message forces the human to investigate; a detailed explanation enables informed response. Check graceful degradation by observing what happens when termination occurs early. Does the agent preserve partial results? Is status clearly reported? Finally, test escalation integration to see whether handoffs work smoothly. When termination requires human judgment, does the human receive sufficient context to decide? A handoff that requires the human to start from scratch is a handoff that failed.

8.8 From Termination to Escalation

Termination defines when agents stop, but not all stopping is the same. Success termination means the task is complete; deliver the results. Failure termination means the task is impossible; report why. Escalation termination means the agent cannot determine success or failure on its own; human judgment is required.

The third category is critical. An agent might complete its search and find conflicting authority, leaving it unable to determine whether the research question has been answered. It might approach a decision that exceeds its authorization. It might recognize that the situation is novel in ways that make its confidence unreliable. In each case, the right response is not to terminate with a result or a failure, but to terminate with a request for human input.

Section 9 examines this closely related question: when should an agent stop autonomous operation and ask for human help? Termination and escalation together define the boundaries of autonomous execution. Without termination, agents run forever. Without escalation, agents exceed authority. These boundaries make agent deployment safe—or at least safer. The following chapter—*How to Govern an Agent*—builds on these termination and escalation mechanisms, providing the governance architecture that enables robust oversight, regulatory compliance, and accountability for agentic system deployments.

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

The best junior associates know when to go to the supervisor. They don't interrupt the partner with every question, but they also don't proceed confidently into territory beyond their expertise. They recognize authority boundaries: "I can draft this motion, but I need partner review before filing." They recognize competence limits: "I've researched for two hours and can't 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 need 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 knowing where to draw 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 want from junior professionals. Agents should do the same.

9.1 When to Escalate

Three categories of triggers warrant escalation: mandatory triggers, confidence-based triggers, and error detection.

Some situations require human involvement regardless of the agent's confidence. Budget exhaustion is a clear mandatory trigger. When the agent approaches resource limits (tokens, time, iterations, cost), it should escalate with a progress summary rather than stopping silently: "I've used 80% of the research budget. Here's what I found. Options: (a) grant additional budget, (b) conclude with current findings, (c) provide strategic guidance on where to focus remaining effort." High-stakes actions require human approval regardless of agent confidence: filing court documents, sending client communications, executing large trades, making regulatory submissions. These are approval gates where the agent prepares and the human authorizes. Authority boundaries trigger escalation when the action exceeds what the agent is authorized to do autonomously; even if the agent is confident in its recommendation, organizational policy may require human sign-off above certain thresholds. Irreversible actions warrant escalation because once you file with the court or execute the trade, you cannot take it back.

Confidence-based escalation occurs when uncertainty about the right answer or approach exceeds acceptable thresholds. Low confidence on output might manifest as: "I found conflicting circuit authority. I'm not confident which rule applies in our jurisdiction" for legal research, or "Correlations have spiked beyond historical norms and model assumptions may be violated" for portfolio analysis. Conflicting information triggers escalation when data sources disagree: the 10-K revenue does not match the earnings release, or two authoritative sources give different answers. Novel situations that do not match patterns the agent has seen warrant human expertise: novel legal questions, unusual market conditions, unprecedented fact patterns. Ambiguous instructions should trigger escalation when, despite clarification attempts (Section 3), the agent remains uncertain about what is being asked.

Error and anomaly detection triggers escalation when something has gone wrong and the agent cannot fix it. Repeated tool failures warrant escalation: if Westlaw times out three times consecutively, the agent should report "Research tool unavailable" with options to wait, use alternatives, or proceed manually. Data anomalies, such as revenue figures that do not reconcile or filing dates that seem wrong or parties that appear on multiple sides of a transaction, warrant human investigation. Constraint violations occur when the task as specified would violate a policy, and the agent should report: "Executing this trade would exceed the position limit. Please confirm override or adjust the order." Impossible requirements should be reported when analysis reveals the task cannot be completed as specified due to conflicting requirements, missing prerequisites, or logical impossibilities.

9.2 How to Escalate

Effective escalation provides the human with everything needed to make a decision. A five-part structure ensures completeness.

The escalation should begin with a situation summary that provides brief context for a human who may not have been following closely. Progress to date explains what has been accomplished and what remains, ensuring the human does not have to start from scratch. The escalation trigger explains why the agent is escalating now and what specific condition triggered the handoff. Information gathered presents what relevant information the agent has found; even if incomplete, partial findings are valuable. Options or recommendations describe the possible paths forward; if the agent has a recommendation, it should state it with supporting reasoning.

Example: Legal Research Escalation.

> **Situation**: Researching statute of limitations for Section 10(b) securities fraud claim.
>
> **Progress**: Searched Westlaw and Lexis. Found clear authority on the 2-year discovery period. Found conflicting circuit authority on when discovery is triggered.
>
> **Trigger**: Low confidence. The Ninth Circuit and Second Circuit apply different tests for inquiry notice. I cannot determine which applies to our facts.
>
> **Findings**: [Summary of key cases with citations]
>
> **Options**: (a) Apply the more conservative test and note the split; (b) research district court authority in our jurisdiction; (c) seek partner guidance on which test likely applies.
>
> **Recommendation**: Option (c): this appears to be a fact-intensive question where partner judgment on the strength of our facts would be valuable.

Example: Financial Escalation.

> **Situation**: Executing rebalancing trades to reduce tech exposure from 35% to 25%.
>
> **Progress**: Generated trade list. Compliance check passed. Ready to execute.
>
> **Trigger**: Trade size exceeds single-approver threshold ($500K total).
>
> **Findings**: Recommended trades would realize $45K in short-term gains and $12K in losses. Net tax impact: approximately $8K additional liability.

Options: (a) Approve full trade list; (b) modify to prioritize tax-loss positions; (c) execute in tranches over multiple days.

Recommendation: Option (a) preferred if reducing exposure is urgent; Option (b) if tax optimization is priority.

9.3 Human-in-the-Loop Patterns

Five patterns integrate human oversight into agent workflows, each suited to different risk profiles and organizational needs.

Approval gates separate preparation from authorization. The agent prepares work product; the human authorizes execution. This pattern is essential for irreversible or high-stakes actions. A litigation agent drafts the court filing, presents it for review, receives approval, then submits. The agent handles preparation; the human controls execution.

Checkpoint reviews provide human verification at milestones to prevent error propagation. A research agent completes legal research, presents authorities, receives confirmation of direction, then proceeds to drafting. Each checkpoint catches misalignment before significant effort is wasted.

Confidence-based escalation ties autonomy to certainty. High confidence triggers autonomous execution; low confidence triggers escalation. A compliance agent processes clear-pass cases automatically while escalating ambiguous situations, balancing efficiency with safety.

The human-as-tool pattern treats human expertise like any other tool. When encountering questions beyond its capabilities, the agent queries the relevant expert, incorporates the response, and proceeds. Human expertise becomes a resource invoked when needed, not a bottleneck for all decisions.

Reversibility classification matches oversight level to action consequences. Fully reversible actions like drafts and research proceed autonomously. Partially reversible actions like communications receive checkpoint review. Irreversible actions like filings and trades require pre-approval. Oversight is proportional to risk.

9.4 Domain-Specific Escalation Requirements

Legal and financial services impose domain-specific escalation requirements rooted in professional responsibility rules and regulatory obligations.

Legal Practice. Legal practice escalation requirements arise from professional responsibility rules:

- **Competence limits**: Escalate when matters exceed agent training or supervising attorney capacity (ABA Model Rule 1.1)
- **Privilege protection**: Escalate any action that might expose privileged information to third parties
- **Conflicts of interest**: Escalate potential conflict situations to conflicts counsel

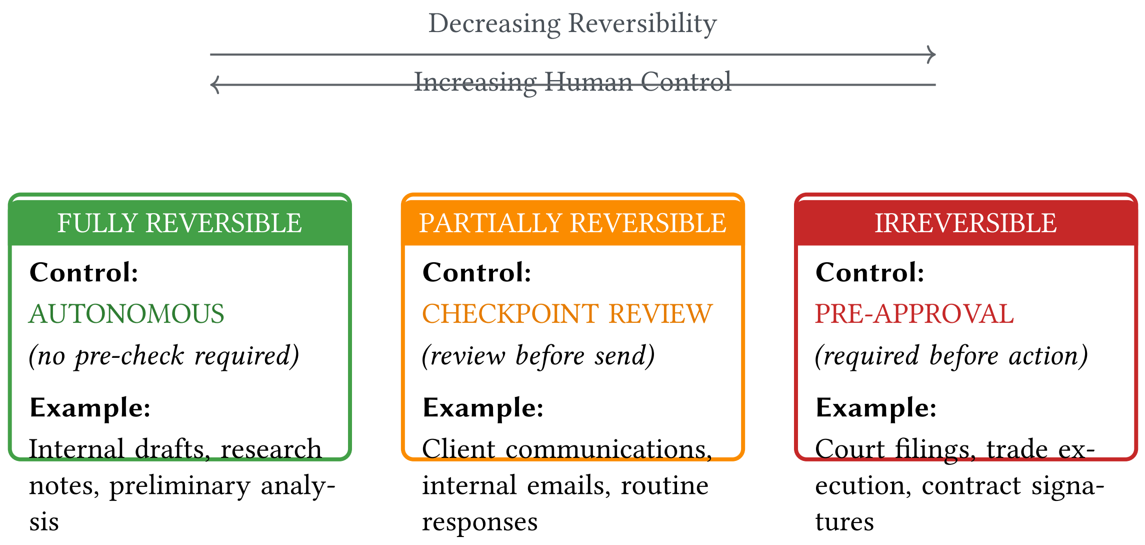


Figure 5: Oversight spectrum showing how the reversibility of agent actions determines the level of human control required. Fully reversible actions may proceed autonomously, partially reversible actions require checkpoint review, and irreversible actions demand pre-approval before execution.

- **Candor to tribunal**: Escalate immediately if adverse authority is discovered that may require disclosure

Financial Services. Financial services escalation requirements arise from regulatory obligations and fiduciary duties:

- **Suitability and fiduciary duty**: Escalate investment recommendations for adviser review before client delivery
- **Regulatory thresholds**: Escalate when trades approach reporting thresholds or disclosure requirements
- **Material non-public information**: Escalate immediately if potential MNPI is encountered
- **Risk limits**: Escalate when proposed actions would breach position limits or risk thresholds

9.5 Evaluating Escalation Mechanisms

When evaluating agentic systems, assess escalation mechanisms against six criteria that distinguish effective systems from those that either escalate too much or too little.

Coverage determines whether all appropriate situations trigger escalation. Test with edge cases: novel situations that should clearly require human judgment, conflicting data that the agent cannot resolve, near-threshold conditions that might slip through. A system with coverage gaps will occasionally proceed autonomously when it should not. Calibration determines whether thresholds are set appropriately. Too sensitive and the agent escalates everything, providing no value; too loose and it proceeds when it should not. Calibrate thresholds against real scenarios, adjusting until the agent escalates when practitioners agree it should.

Latency determines how quickly escalation reaches the right human. For urgent matters—a margin call, a filing deadline, a client emergency—escalation must be immediate. For routine matters, queued escalation may suffice. Routing determines whether escalation reaches the right person. Complex legal questions should reach senior attorneys, not paralegals; risk limit breaches should reach risk managers, not operations staff. Misrouted escalation wastes time and may produce inadequate responses.

Context quality determines whether the human can actually decide. Test by reviewing escalation messages and asking: could you make a decision from this information alone? If the human must investigate further before responding, the escalation is incomplete. Response handling determines whether the agent correctly incorporates human guidance. Test the full cycle, not just escalation initiation; an agent that escalates well but ignores responses provides only the illusion of human oversight.

9.6 From Escalation to Delegation

Escalation moves control *up*: from agent to human supervisor. But agents can also move control *sideways* by delegating subtasks to other agents. Where escalation says "I need human help," delegation says "I need specialist help."

Section 10 examines the next question: how does an agent work with other agents? Delegation patterns enable complex workflows where multiple specialized agents collaborate, each with its own escalation paths back to human oversight. A coordinating agent might delegate research to a legal research specialist, analysis to a financial modeling specialist, and drafting to a document generation specialist—while each specialist retains the ability to escalate to humans when it reaches its own limits.

The combination of escalation (vertical) and delegation (horizontal) defines the full topology of human-agent collaboration. Escalation ensures human oversight. Delegation enables specialization and scale. Together, they make complex agentic workflows possible while maintaining the human control that regulated professions require.

10 How Does an Agent Work with Other Agents?

Complex matters require coordination. The M&A partner does not do everything personally but instead coordinates specialists: corporate counsel reviews governance documents, tax specialists analyze structure, antitrust counsel assesses regulatory risk, employment lawyers review executive agreements. Each specialist has deep expertise in their domain. The partner's job is orchestration: defining what each specialist should produce, ensuring deliverables integrate coherently, and synthesizing conclusions for the client.

A portfolio manager coordinates similarly: research analysts provide company-specific analysis, traders handle execution, risk managers monitor exposure, compliance officers verify regulatory adherence. Complex trades require all these perspectives; no single person has all the 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 in narrow domains, orchestrators that coordinate their work, and protocols that enable 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.

Specialization allows each agent to excel in a narrow domain. A securities law agent can be optimized for SEC regulations, loaded with relevant precedent, and equipped with EDGAR tools. A separate tax agent handles tax implications with different tools and knowledge. Neither needs to be expert in the other's domain.

Security isolation gives each agent minimum necessary permissions. The research agent can read legal databases but cannot file documents. The 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. The document review agent analyzes contracts while the research agent investigates legal issues. Neither waits for the other. Vendor diversity allows different agents to use different models or providers: a specialized legal model for research, a general model for drafting, a fast model for classification. Multi-agent enables best-of-breed selection.

Scale management addresses context window limits. Rather than cramming everything into one context, you can decompose across agents, each with focused context.

The tradeoffs are coordination overhead (agents must communicate), debugging complexity (failures span multiple agents), and additional security surface (more agents means more potential attack vectors).

10.2 Agent-to-Agent Protocol (A2A)

The Agent-to-Agent Protocol standardizes how agents collaborate, complementing MCP's tool integration. If MCP is how agents access resources, A2A is how agents delegate work to specialists.

A2A uses familiar professional concepts. Agent Cards are capability statements, like a specialist's résumé listing expertise, input requirements, and output formats. Before delegating, the coordinator reviews the Agent Card to confirm the specialist can handle the task.

Tasks are units of delegated work, like engagement letters specifying scope, constraints, and deadlines. The coordinator creates a Task; the specialist accepts and executes. Upon completion, specialists return Artifacts: draft memos, analysis reports, structured data. The coordinator integrates these work products into the final deliverable.

Communication Channels support asynchronous, long-running work. You assign research Monday; the memo arrives Friday. Channels enable status updates and clarification requests during execution.

Agent collaboration follows five phases mirroring professional delegation:

A2A Task Delegation

The task lifecycle begins with **discovery**: finding the right specialist via Agent Card, much like finding co-counsel through a directory. Next comes **delegation**: creating a Task with goals, constraints, and deadline, analogous to drafting an engagement letter. During **execution**, the specialist works independently and may request clarification, like an associate researching a legal question. Upon **delivery**, the specialist returns Artifacts, submitting a draft memo for review. Finally, **completion** occurs when the coordinator reviews, approves, or requests revision—the partner review that ensures quality.

The key insight is that A2A enables delegation without micromanagement: you define *what* needs to be done; the specialist decides *how*.

10.3 Multi-Agent Patterns

Three patterns organize multi-agent collaboration.

In sequential delegation, specialists work in sequence, each building on prior work: the Coordinator delegates to a Research Agent, whose output flows to an Analysis Agent, then to a Drafting Agent, before the Coordinator synthesizes the final result. This pattern is simple but slow because each specialist waits for prior completion.

Parallel delegation allows independent specialists to work simultaneously. Securities, Tax, and Employment Agents can analyze an acquisition concurrently while the Coordinator integrates their findings. This approach is faster but requires task independence.

Hierarchical delegation enables specialists to delegate to sub-specialists. A Lead Due Diligence Agent might delegate to Document Review and Legal Research agents, each of which may further delegate. This pattern handles complex matters but introduces coordination overhead and cascading failure risk.

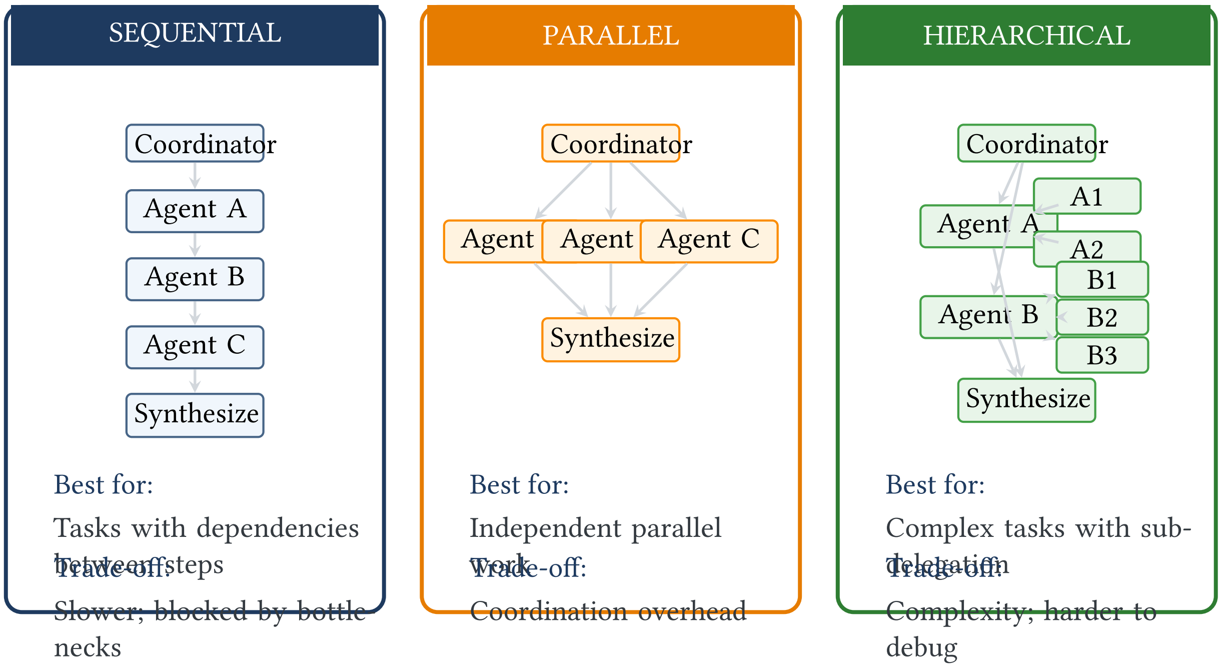


Figure 6: Three multi-agent orchestration patterns. Sequential delegation chains agents in order, ideal for dependent tasks but vulnerable to bottlenecks. Parallel delegation runs agents concurrently, maximizing throughput for independent work but requiring coordination. Hierarchical delegation enables sub-agents to handle specialized sub-tasks, providing flexibility for complex workflows at the cost of debugging complexity.

Production systems typically require both MCP and A2A working in concert. MCP handles agent-to-tool communication: each specialist agent uses MCP to access its domain-specific tools, including databases, calculators, and external systems. A2A handles agent-to-agent coordination: the orchestrator uses A2A to delegate tasks to specialists and receive artifacts.

Consider an M&A due diligence example. The orchestrator receives the instruction to conduct due diligence on a target company acquisition and delegates via A2A to three specialists. The Document Processing Agent indexes the data room, classifies documents, and extracts key terms, using MCP to access document management systems and OCR tools. The Financial Analysis Agent analyzes financial statements, builds models, and identifies risks, using MCP to access financial databases and calculation tools. The Legal Risk Agent reviews contracts for problematic provisions and researches legal issues, using MCP to access legal research platforms and precedent databases.

Each specialist returns Artifacts via A2A. The orchestrator synthesizes these into a comprehensive due diligence report. Throughout, MCP handles tool access while A2A handles coordination. Neither protocol alone suffices.

10.4 Multi-Agent Workflow Examples

Consider a legal workflow for regulatory compliance assessment. A fintech company asks what regulatory approvals it needs to launch a new product. The Orchestrator receives this query and decomposes it into regulatory domains.

The Securities Agent analyzes whether the product involves securities, using MCP to search SEC guidance, no-action letters, and case law. It concludes that the product likely constitutes a security under the Howey test and that registration or an exemption is required. The Banking Agent analyzes banking law implications, searching OCC, FDIC, and state banking guidance. It determines that no bank charter is required but that state money transmitter licenses may apply.

The Consumer Protection Agent analyzes CFPB jurisdiction and state consumer laws, identifying UDAP exposure and recommending clear disclosures and complaint handling procedures. The AML Agent analyzes Bank Secrecy Act obligations and determines that FinCEN registration is required along with a KYC/AML program.

The Orchestrator synthesizes these findings into a comprehensive regulatory roadmap with prioritized action items.

A financial workflow illustrates large block trade execution. A portfolio manager requests execution of a $50M block trade in XYZ Corp while minimizing market impact. The Orchestrator decomposes this into analysis and execution phases.

The Market Agent analyzes current liquidity, trading patterns, and optimal execution windows using MCP for market data feeds. It recommends VWAP execution over two days given average daily volume of $200M. The Compliance Agent verifies that the trade does not breach limits or create reporting obligations, using MCP for compliance databases. It confirms the trade is within limits but notes that a 13F amendment will be required at quarter-end.

The Risk Agent assesses the impact on portfolio risk metrics using MCP for the risk engine. It finds that the trade increases sector concentration by 2%, which remains within policy limits. The Execution Agent implements the execution strategy using MCP for order management systems and confirms that the order has been placed.

The Orchestrator monitors execution, coordinates any necessary adjustments, and reports completion to the portfolio manager.

10.5 Multi-Agent Risks

Multi-agent systems introduce failure modes and security challenges beyond single-agent problems.

Several coordination failure patterns can emerge. Deadlock occurs when agents wait for each other cyclically and neither proceeds; prevention requires clear task dependencies, timeouts, and circular-wait detection. Divergent conclusions arise when specialists reach incompatible results; the orchestrator must detect conflicts and either reconcile them or escalate to human judgment.

Cascading errors occur when incorrect output from one agent propagates through dependent agents. Prevention requires validating inputs at each handoff rather than trusting upstream agents. Coordination overhead presents a different challenge: communication consumes tokens, time, and cost, so for simple tasks, the overhead may exceed specialization benefits. Use multi-agent architectures when complexity justifies the coordination cost.

Accountability gaps emerge when something fails and it is unclear which agent is responsible. Prevention requires comprehensive logging at every delegation, clear audit trails, and defined accountability for each subtask.

Multi-agent systems also require security controls at the coordination layer. Agent identity is fundamental: each agent must have verifiable identity, with cryptographic signatures authenticating Agent Cards and task responses to prevent impersonation attacks.

Authorization controls restrict delegation relationships. Not every agent can delegate to every specialist; access control policies define which delegation paths are valid. Information barriers enforce legal conflicts and financial Chinese walls across agent boundaries. An agent working on Company A's acquisition cannot delegate to agents with access to Company A's confidential information from other engagements.

Audit trails must capture every delegation: who delegated what to whom, when, with what constraints, and what was returned. This enables post-hoc analysis and compliance review. Finally, task validation requires specialists to verify that tasks fall within their authorized scope and to reject tasks that would require accessing forbidden data or taking unauthorized actions.

10.6 Protocol Selection Guidance

Table 4: Protocol selection cues

Signal	**Protocol**	**Latency**	**Examples**
Immediate, well-defined operation	MCP	ms–seconds	Query database; retrieve document; run calculation
Delegated work requiring judgment	A2A	minutes–hours	Assign research; request analysis; coordinate specialists
End-to-end workflow with both	MCP + A2A	blended	Due diligence; portfolio rebalancing; regulatory assessment

Use MCP when you need immediate tool access: database queries, document retrieval, and calculations. Use A2A when you need to delegate work that requires judgment, iteration, or extended execution time. Use both when complex workflows combine tool access (each specialist uses MCP) with coordination (the orchestrator uses A2A).

As of late 2025, MCP is production-ready for tool integration. A2A is maturing with a stable specification and active pilots, but cross-vendor reliability remains uneven. Design fallbacks to human coordination where A2A would ideally apply.

10.7 From Delegation to Governance

Delegation distributes work across agents, creating governance challenges single-agent systems avoid. Accountability becomes complex: when a multi-agent workflow errs, responsibility could lie with the coordinator, the specialist, or the human who approved output. Information barriers applying to human teams must apply to their agents—an agent cannot access data its human principal could not. Audit trails must span the entire delegation tree; when regulators ask what happened, the trail must show every delegation, handoff, and decision point.

Section 11 previews governance requirements including accountability models and audit architecture.

11 How Do We Keep the Agent Safe?

Every professional organization has compliance programs, audit functions, and oversight structures. The law firm has conflicts committees, billing review, and quality control. The financial institution has risk management, compliance monitoring, and internal audit. These functions do not do the work themselves; they ensure the work is done safely, ethically, and in compliance with applicable rules.

Agentic systems require the same infrastructure. Governance is not a single question but a lens through which all other questions must be viewed. Every capability creates governance requirements. Every architectural choice enables or constrains oversight. This section previews governance across all ten questions; the following chapter explores these requirements in depth.

Agentic System Governance

Agentic system governance encompasses the policies, controls, and oversight mechanisms that ensure agentic systems operate safely, ethically, and in compliance with applicable requirements. Governance spans the agentic system lifecycle: design, deployment, operation, and retirement.

Governance is not optional for regulated professional services. Professional duties are non-delegable: attorneys remain liable for AI-assisted work product, and fiduciaries remain accountable for AI-informed recommendations.

11.1 Architecture Enables Governance

The architectural choices throughout this chapter are not merely technical decisions. They are the *infrastructure* that makes governance possible.

You cannot audit what you did not log. You cannot enforce privilege boundaries that were never implemented. You cannot demonstrate bounded operation without termination mechanisms. When a regulator asks how the compliance agent detected a breach, when opposing counsel demands production of the agent's reasoning, when a client questions why the agent recommended a particular strategy—architecture determines whether you can answer.

Professional duties are non-delegable: attorneys remain liable for AI-assisted work product, and fiduciaries remain accountable for AI-informed recommendations. The following chapter details those obligations. This chapter provides the architecture to meet them. Section 13.1 provides a comprehensive mapping of architectural choices to governance implications.

11.2 Governance Requirements by Question

Each of the ten questions creates specific governance requirements:

11.3 Security Essentials

Five security controls are essential for any agentic system deployment in regulated contexts:

Security Controls for Regulated Practice

1. **Input separation**: Isolate user inputs from system prompts to prevent prompt injection attacks
2. **Output validation**: Verify agent outputs before execution to detect hallucinations and constraint violations
3. **Least privilege**: Grant minimum necessary tool access to limit the scope and impact of failures
4. **Audit logging**: Maintain comprehensive action logs for accountability and investigation
5. **Matter/client isolation**: Enforce confidentiality boundaries to protect privileged and confidential information

These controls map to the ten-question framework:

- Input separation protects Q2 (Intent) from manipulation
- Output validation governs Q4 (Action)

Table 5: Ten-question governance mapping

Q	Question	Governance Requirement
1	Triggers	Event authorization, audit logging of all triggers
2	Intent	Purpose limitation, goal alignment verification
3	Perception	Data governance, access controls, provenance tracking
4	Action	Actuation controls, approval gates, rollback capability
5	Memory	State integrity, retention policies, isolation enforcement
6	Planning	Bounded operation, resource budgets, plan validation
7	Termination	Exit protocols, success criteria, graceful degradation
8	Escalation	Human oversight, escalation triggers, response tracking
9	Delegation	Agent identity, delegation contracts, barrier enforcement
10	Governance	Meta-governance, audit architecture, compliance monitoring

- Least privilege limits Q3 (Perception) and Q4 (Action)
- Audit logging enables Q7 (Termination) review and Q8 (Escalation) tracking
- Matter/client isolation enforces Q5 (Memory) boundaries

11.4 Transparency and Explainability

Regulators and clients increasingly require explanations for agentic system decisions. Four levels of transparency serve different audiences, illustrated here with a breach detection agentic system:

Level 0 provides output only: just the answer, such as “Suspicious transaction flagged.” This level suffices for routine, low-stakes queries where the consumer trusts the system.

Level 1 adds a summary with sources: the conclusion plus citations, such as “Transaction #45921 flagged; exceeds threshold in Rule 203(b)(1).” This enables verification without requiring full reasoning.

Level 2 provides a reasoning outline: key analytical steps plus sources, such as “Flagged because: (1) $150K exceeds $100K threshold, (2) counterparty on watchlist, (3) timing matches known pattern.” This level is appropriate for substantive work product requiring review.

Level 3 provides a full execution trace: a structured record of tool calls, retrieved documents, and decision points, including database queries, rule evaluation steps, and confidence scores. This level enables audit and debugging.

The architecture should capture Level 3 traces for all operations, then generate audience-appropriate summaries (Levels 0–2) on demand.

11.5 Auditability vs. Retention

A tension exists between comprehensive logging (for audit) and data minimization (for privacy and compliance). The resolution is *not* “log everything forever.” Instead, four practices balance these competing demands.

Structured logging captures structured decisions rather than raw chain-of-thought. Structure enables selective retention because decision points can be preserved while ephemeral reasoning is discarded. Tiered retention implements different periods for different purposes: short-term operational logs with full detail retained for days to weeks, medium-term audit logs with structured decisions retained for months to years, and long-term compliance archives containing minimal but sufficient information as required by regulation.

Redaction at capture applies privacy and confidentiality filters before logging, not after; once sensitive information enters logs, it becomes difficult to remove systematically. Finally, legal hold integration ensures that retention schedules yield to preservation obligations when litigation is anticipated.

The following chapter provides detailed retention frameworks for legal and financial contexts.

11.6 Forward to Chapter 8

This chapter answered *how to build an agentic system.* The ten questions (Table 5) provide architectural foundations; each creates governance requirements that the architecture must support.

The following chapter answers: *how do we govern these systems?* It examines the five-layer governance stack (legal, model, system, process, culture), dimensional controls (autonomy, persistence, goal dynamics), accountability structures, and regulatory compliance frameworks—including Federal Reserve model risk management guidance (Board of Governors of the Federal Reserve System and Office of the Comptroller of the Currency 2011) and the NIST AI Risk Management Framework (National Institute of Standards and Technology 2023)—with worked examples in legal, financial, and audit contexts.

Architecture provides the foundation; governance provides the controls. Together, they enable responsible deployment of agentic systems in regulated professional services.

12 Synthesis: Reference Architectures

The previous sections examined each of the ten questions in isolation. This section shows how they work together in complete deployments. Two reference architectures demonstrate the full framework while honestly acknowledging current limitations: one legal, one financial.

Reference Architectures, Not Production Claims

These architectures illustrate how components fit together as *design targets*, not claims about current reliability. The reliability cliff (Section 8.5) constrains what agentic systems achieve today; these multi-hour workflows require extensive human oversight. Read as "how you would design it" not "how it works today."

12.1 Case Study: Credit Facility Documentation Review

A corporate client is borrowing $500 million under a senior secured revolving credit facility. The law firm represents the borrower. The partner assigns the matter: "Review the draft credit agreement and identify provisions that differ materially from market terms or our standard positions."

This is a document review task that would traditionally require 8–12 hours of senior associate time. The goal is not to replace the associate but to accelerate the review and ensure comprehensive coverage. The ten-question framework guides every design choice.

Q1 (Triggers): Work enters via document upload to the deal room and partner assignment through the matter management system. The trigger is explicit: new document plus assignment.

Q2 (Intent): The agentic system extracts intent: document review task, borrower perspective, focus on material deviations from market and template. Implicit constraints include confidentiality (privileged work product) and deadline (closing in two weeks).

Q3 (Perception): The agentic system uses MCP Resources to access the draft agreement (document management), the firm's template facility agreement (precedent database), and market terms data (external database). Read-only access; no modifications.

Q4 (Action): Action tools are limited to document annotation (internal markup) and memo generation (work product creation). No external actions: filing, communication, or execution.

Q5 (Memory): Episodic memory tracks analysis progress (which sections reviewed, what issues identified). RAG provides access to prior credit agreement memos and deal histories. Matter isolation ensures this work doesn't access unrelated client information.

Q6 (Planning): Plan-Execute pattern. The agentic system creates a section-by-section review plan based on the table of contents. Systematic execution through financial covenants, events of default, representations, conditions precedent.

Q7 (Termination): Success criteria: all material sections reviewed, issues identified and categorized, draft memo produced. Budget: token limit, time limit, iteration limit. Checkpoint after initial scan to confirm scope.

Q8 (Escalation): Escalate on: ambiguous provisions requiring legal judgment, potential conflicts with other client matters, issues that might affect deal viability. Human-as-tool pattern for partner input on materiality thresholds.

Q9 (Delegation): Single-agent architecture for this task. Multi-agent would be appropriate if combined with separate research (legal issues) or financial modeling (covenant analysis) workstreams.

Q10 (Governance): Audit logging of all document access and analysis steps. Privilege protection enforced. Work product clearly marked as AI-assisted for attorney review.

The agentic system proceeds systematically through the review process:

1. Partner assigns matter; agentic system receives trigger
2. Agentic system retrieves credit agreement, template, market terms
3. Agentic system creates review plan: 15 sections, estimated analysis per section
4. Agentic system analyzes Section 1 (Definitions): identifies unusual defined terms, compares to template
5. Agentic system continues through financial covenants: flags leverage ratio that differs from market
6. Agentic system reviews events of default: identifies cross-default threshold below typical market terms
7. ... [continues through all sections]
8. Agentic system compiles findings into issues list and draft memo

9. Agentic system presents to associate for review
10. Associate reviews, adds context, escalates significant issues to partner

Even well-designed agentic systems fail. Understanding failure modes is crucial. The agentic system may miss nuanced definitions where a defined term has been subtly modified from the template in ways that affect covenant calculations. Human review catches subtleties like: "EBITDA is defined to exclude one-time charges, but the add-back is capped—that's unusual and limits flexibility."

Cross-document dependencies present another risk. The credit agreement references schedules and exhibits; if the agentic system does not trace these references and analyze the schedules, material issues may be missed. Market context adds complexity because "market terms" vary by borrower credit quality, industry, and timing. The agentic system compares to a template, but the template may not reflect current market conditions for this borrower's profile.

The most dangerous failures are omissions: things the agentic system does not flag because it does not recognize their significance. Human expertise identifies gaps like: "There's no limitation on amendments to subordinated debt—that's a significant gap."

Mitigation requires checkpoint review after initial scan. The associate reviews agentic system output not just for correctness but for completeness. Partner review of final work product ensures quality. The agentic system accelerates but does not replace human judgment.

12.2 Case Study: Equity Portfolio Management

An investment adviser manages a $200 million equity portfolio for institutional clients. The portfolio manager wants continuous monitoring with agent assistance for rebalancing analysis, compliance checking, and research synthesis.

This is a continuous monitoring and advisory task, not a one-time analysis. The goal is to augment the PM's capacity, not to trade autonomously. The ten-question framework shapes this more complex, multi-agent architecture.

Q1 (Triggers): Multiple trigger types: market data feeds (price movements, earnings releases), scheduled jobs (daily compliance check, weekly rebalancing analysis), human prompts (PM requests analysis), escalation events (position approaching limits).

Q2 (Intent): Intent varies by trigger. Price alert: assess significance and recommend action. Scheduled rebalance: generate trade list if allocations drift beyond thresholds. PM query: answer specific question about position or strategy.

Q3 (Perception): MCP Resources access market data (prices, fundamentals), portfolio data (positions, P&L), research (analyst reports, news), and compliance data (client guidelines, regulatory limits). Read-only access to trading systems.

Q4 (Action): The agentic system can generate recommendations and create reports. Trade execution requires PM approval—the execution action tool is behind an approval gate. No autonomous trading.

Q5 (Memory): Episodic memory tracks recent analysis, PM decisions, and rationales. RAG accesses investment research archive. Client isolation ensures each client's portfolio data is segregated.

Q6 (Planning): ReAct pattern for ad hoc analysis (exploratory). Plan-Execute for scheduled tasks (systematic). Hierarchical for comprehensive reviews (decompose to specialists).

Q7 (Termination): Varies by task. Monitoring: continuous (no termination). Analysis: complete when question answered. Rebalancing: complete when trade list generated and approved.

Q8 (Escalation): Escalate on: positions approaching limits, unusual market conditions, conflicting signals, any trade recommendation (PM approval required). Risk management escalation path for limit breaches.

Q9 (Delegation): Multi-agent architecture. Monitoring agent watches market data. Research agent synthesizes analyst reports. Compliance agent checks guidelines. Rebalancing agent generates trade recommendations. PM agent orchestrates and presents to human PM.

Q10 (Governance): Comprehensive audit trail. Fiduciary duty documentation (rationale for recommendations). Compliance monitoring. MNPI controls (no access to deal team information).

Multiple agents coordinate to generate and validate recommendations:

1. Monitoring agent detects: "Tech sector up 3% today; portfolio tech allocation now 35% vs. 25% target"
2. Monitoring agent triggers Rebalancing agent
3. Rebalancing agent queries current positions (MCP)
4. Rebalancing agent generates trade options: sell $20M tech, options include [specific positions]
5. Rebalancing agent queries Compliance agent: "Check proposed trades against guidelines"
6. Compliance agent confirms: trades within limits, no restricted securities
7. Rebalancing agent queries Research agent: "Any recent negative research on proposed sales?"
8. Research agent returns: "No material negative research; one position has earnings next week"
9. Rebalancing agent adjusts: defer one sale until after earnings
10. Rebalancing agent presents recommendation to PM agent
11. PM agent formats for human review, highlights key considerations
12. Human PM reviews, approves (or modifies), authorizes execution
13. Execution agent (with PM approval) places orders via OMS

Coordination introduces additional failure vectors beyond single-agent systems.

Cascading errors occur when the Monitoring agent misinterprets data, the Rebalancing agent acts on a bad signal, and trades are recommended that should not be. Mitigation requires validation at each handoff and sanity checks on data before acting. Coordination overhead presents a different challenge: communication between agents consumes tokens and time, so for simple decisions, the overhead may exceed the value. Monitor coordination costs and simplify the architecture when overhead dominates.

Debugging complexity increases with multi-agent systems. When recommendations are wrong, tracing the error through multiple agents is difficult. Mitigation requires comprehensive logging, clear attribution at each step, and replay capability. Agent disagreement occurs when the Research agent sees a positive signal while the Risk agent sees a negative one. Clear escalation protocols determine what happens when agents conflict; humans resolve material disagreements.

12.3 Synthesis: Principles Across Domains

Both case studies illustrate common principles that apply across legal and financial agent deployments.

Decomposition is essential. Neither workflow attempts end-to-end autonomous completion. The credit facility review breaks into 15 section-by-section analyses rather than attempting one pass; the portfolio rebalancing separates constraint checking from trade generation from execution. Tasks are decomposed into manageable steps with human checkpoints.

Human-in-the-loop oversight is the norm. Both architectures assume human oversight at critical junctures: attorney review of legal analysis before client delivery, PM approval of trade recommendations before execution. Agents augment professional judgment; they do not replace it.

Memory enables continuity. Both rely on episodic memory (what happened in this session) and RAG (institutional knowledge). The legal agentic system retrieves prior firm precedent on similar provisions; the financial agentic system accesses historical rebalancing decisions. Without memory, every interaction starts fresh.

Isolation is non-negotiable. Matter isolation (legal) and client isolation (financial) are architectural requirements, not optional features. The agentic system working on Company A's financing cannot access Company B's confidential terms, even if both are firm clients.

Failure modes are predictable. The same failure patterns appear in both domains: nuanced judgment that exceeds current capabilities, omissions where agentic systems miss non-obvious issues, and cascading errors where early mistakes propagate through analysis. Design for these failures explicitly.

Governance is pervasive. Every architectural choice has governance implications. Audit trails document what the agentic system accessed and concluded; approval gates ensure human review before irreversible actions; escalation paths route uncertainty to appropriate decision-makers. These reference architectures establish the *capability* foundations; the following chapter—*How to Govern an Agent*—provides the *control* frameworks: the five-layer governance stack, dimensional calibration, regulatory compliance structures, and accountability mechanisms required for deployment in regulated industries.

Current limitations are real. Neither architecture claims autonomous completion of multi-hour tasks. The reliability cliff constrains what these agentic systems can do today; design accordingly.

12.4 Framework Completion Checklist

Before deploying any agentic system, verify that all ten questions have been answered:

Ten-Question Deployment Checklist

- ☐ **Triggers**: How does work enter? Are all trigger types covered? Is there audit logging?
- ☐ **Intent**: How is intent extracted? What happens with ambiguity? Are constraints identified?
- ☐ **Perception**: What tools provide information? Is access properly controlled? Is provenance tracked?
- ☐ **Action**: What can the agentic system do? Are irreversible actions gated? Is rollback possible?
- ☐ **Memory**: What persists across sessions? Is isolation enforced? What are retention policies?
- ☐ **Planning**: What pattern applies? Are budgets enforced? Is there loop detection?
- ☐ **Termination**: How does the agentic system know when it's done? What are success criteria? How does it handle failure?
- ☐ **Escalation**: When does the agentic system ask for help? Who receives escalations? Is context sufficient?
- ☐ **Delegation**: Does it coordinate with other agents? Are protocols standardized? Are barriers enforced?
- ☐ **Governance**: Are security controls implemented? Is there audit capability? Are professional duties met?

Any question left unanswered represents a gap in the architecture that will manifest as a failure in production.

13 Conclusion: From Architecture to Governance

This chapter opened with a claim: **agents are not magic; they are architecture**. Ten sections later, that claim should feel concrete.

You now know how work reaches an agent (triggers and channels), how instructions become goals (intent extraction), how agents gather information (perception tools) and take action (action tools with approval gates), how context persists across sessions (memory architecture), how complex work decomposes into manageable steps (planning patterns), how agents recognize completion (termination criteria), when they hand off to humans (escalation logic), how they coordinate with other agents (delegation protocols), and what controls keep them safe (governance infrastructure).

Each of these capabilities corresponds to design decisions with real tradeoffs. Richer memory improves context but increases cost and latency. More aggressive escalation improves safety but reduces autonomy. Tighter approval gates reduce risk but slow execution. There are no free lunches in agent architecture, only tradeoffs that must be calibrated to your specific context, risk tolerance, and regulatory environment.

The organizational analogy that structured this chapter is not merely pedagogical. Law firms and investment teams are cognitive work systems that have evolved sophisticated infrastructure for exactly the challenges agents face: distributing cognitive work, maintaining context, coordinating specialists, ensuring quality, and enabling oversight. When you evaluate an agentic system, you can ask the same questions you would ask about a professional team. When you design governance, you can draw on the same frameworks.

13.1 What This Understanding Enables

With architectural literacy, you can evaluate vendor claims critically. When a vendor says their agent "handles legal research," you know to ask: What triggers initiate research? How does it understand the research question? What databases does it access? How does it know when research is complete? What happens when confidence is low? The ten questions provide a systematic evaluation framework.

You can also participate meaningfully in procurement. You can assess whether a proposed system meets your organization's requirements. Does it enforce matter isolation? Does it maintain audit trails? Does it integrate with your approval workflows? Can it escalate appropriately? You have vocabulary to specify requirements in terms developers understand.

This understanding lets you design governance that maps to architecture. Governance is not separate from architecture—it is enabled by architecture. If you want audit trails, the system must log its reasoning. If you want approval gates, the system must pause before consequential actions. If you want confidentiality, the system must isolate matter context. You can design systems where governance is built in, not bolted on.

Finally, you can communicate with technical teams. You can describe requirements precisely: "I need perception tools for these three databases, action tools behind approval gates for these two operations, escalation triggers when confidence drops below threshold, and memory isolation between client matters." Shared vocabulary enables collaboration.

13.2 Honest Assessment of Current Capabilities

Architectural understanding should include honest acknowledgment of current limitations.

The reliability cliff is perhaps the most significant constraint. As discussed in Section 8.5, agents exhibit near-perfect success on tasks under four minutes but under 10% success on tasks exceeding four hours. This is not a minor limitation—it fundamentally shapes what agents can reliably do today. Design systems that decompose work aggressively, that checkpoint progress frequently, and that escalate before reliability degrades.

Judgment limitations constrain where agents add value. Agents excel at retrieval, systematic execution, and pattern matching, but they struggle with nuanced judgment, novel situations, and tasks requiring deep domain expertise. The most effective deployments pair agent capabilities with human judgment rather than attempting to replace it.

Brittleness causes production systems to fail unpredictably due to API changes, authentication expiration, format variations, and edge cases that deterministic systems would handle gracefully. Build monitoring, alerting, and graceful degradation into every production deployment.

Compounding errors affect multi-step workflows. Error probabilities multiply at each step, so a workflow with ten steps that are each 95% reliable succeeds only 60% of the time end-to-end. Shorter workflows with human checkpoints outperform longer autonomous chains.

These limitations are not permanent, but they are real today. Design systems that deliver value despite limitations, not systems that assume limitations away.

13.3 Essential Resources

For practitioners moving from concepts to deployment, four resource categories provide essential guidance.

Security fundamentals should be addressed before any production deployment. Implement the five controls from Section 11.3: input separation, output validation, least privilege, audit logging, and matter/client isolation. The OWASP LLM Top 10 provides vulnerability taxonomy; the NIST AI Risk Management Framework offers lifecycle guidance.

Protocols and standards define interoperability. The Model Context Protocol (MCP) standardizes agent-to-tool communication and is production-ready with thousands of available servers. The Agent-to-Agent Protocol (A2A) standardizes multi-agent coordination and is maturing under the Linux Foundation.

Research foundations provide theoretical grounding. Key works include Xi et al. on agent architecture (Xi et al. 2023), Yao et al. on ReAct (Yao et al. 2022), and Park et al. on memory (Park et al. 2023). For evaluation, LegalBench addresses legal reasoning (Guha et al. 2023) and VLAIR measures legal AI performance against lawyer baselines (Vals AI 2025).

Regulatory guidance varies by domain. Legal practitioners should monitor ABA ethics opinions, particularly Formal Opinion 512 on supervision (American Bar Association Standing Committee on Ethics and Professional Responsibility 2024). Financial practitioners should monitor SEC guidance, FINRA communications, and prudential regulators on model risk management.

Temporal Warning

Resources accurate as of late 2025 may not reflect subsequent developments. Protocol specifications evolve, regulatory frameworks develop, security vulnerabilities emerge. Verify currency before relying on any reference for production decisions.

13.4 From Architecture to Governance

This chapter answered: *How do you build an agent?*

The next chapter answers: *How do you govern one?*

These questions are deeply connected. Every architectural decision in this chapter has governance implications:

- Trigger logging creates audit trails of what initiated agent action
- Intent extraction allows review of what the agent understood
- Perception controls enforce data governance and access restrictions
- Action gates require approval workflows and human oversight
- Memory isolation protects confidentiality and privilege
- Planning budgets ensure bounded, predictable operation
- Termination criteria verify task completion
- Escalation paths route uncertainty to human judgment
- Delegation contracts establish accountability across agent teams

The relationship is not incidental. **Architecture enables governance.** If you did not architect for logging, you cannot audit. If you did not architect for approval gates, you cannot require sign-off. If you did not architect for isolation, you cannot enforce confidentiality boundaries. Governance is not bolted onto architecture after the fact; it emerges from architectural decisions made at design time.

The next chapter builds on this foundation. Where this chapter focused on *capability*—what agents can do and how they do it—the next focuses on *control*—ensuring agents do what they should, only what they should, and nothing they should not. The governance frameworks, oversight mechanisms, and compliance strategies in that chapter assume the architectural understanding developed here.

You cannot govern what you do not understand.

You cannot control what you did not architect.

Now you can do both.

References

Administrative Office of the U.S. Courts (2024a). *Electronic Filing (CM/ECF)*. Case Management/Electronic Case Files system for filing federal court documents online; mandatory for most federal court filings. URL: https://www.uscourts.gov/court-records/electronic-filing-cm-ecf (visited on 12/13/2025).

Administrative Office of the U.S. Courts (2024b). *Public Access to Court Electronic Records (PACER)*. Official PACER system providing electronic public access to federal appellate, district, and bankruptcy court records. URL: https://pacer.uscourts.gov/ (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).

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).

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).

Guha, Neel, Julian Nyarko, Daniel E. Ho, Christopher Ré, Adam Chilton, Alex Chohlas-Wood, Austin Peters, Brandon Walber, Nika Haghtalab, et al. (2023). "LegalBench: A Collaboratively Built Benchmark for Measuring Legal Reasoning in Large Language Models". In: *arXiv preprint arXiv:2308.11462*. 162 tasks from 40 contributors covering six types of legal reasoning; developed by Stanford and HazyResearch.

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).

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).

National Institute of Standards and Technology (Jan. 2023). *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*. Tech. rep. NIST AI 100-1. Voluntary framework for managing AI risks; organized around governance, mapping, measuring, and managing AI risks; referenced by EU AI Act and other regulatory frameworks. U.S. Department of Commerce. URL: https://www.nist.gov/itl/ai-risk-management-framework (visited on 12/13/2025).

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).

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.

Reimers, Nils and Iryna Gurevych (2019). "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Foundational work on sentence embeddings for semantic similarity and retrieval; basis for modern vector search in RAG systems. Association for Computational Linguistics. URL: https://arxiv.org/abs/1908.10084 (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).

U.S. Securities and Exchange Commission (2024). *About EDGAR*. Electronic Data Gathering, Analysis, and Retrieval system for SEC filings; provides free public access to company filings. URL: https://www.sec.gov/submit-filings/about-edgar (visited on 12/13/2025).

Vals AI (Oct. 2025). *VLAIR: A Benchmark for Evaluating Legal AI Against Lawyers*. First benchmark comparing legal AI systems against lawyer control groups across seven tasks; AI scored 7 points above lawyer baseline (71% accuracy), outperforming on routine tasks but underperforming on complex judgment-intensive work. URL: https://www.vals.ai/vlair (visited on 12/13/2025).

Xi, Zhiheng, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. (2023). "The Rise and Potential of Large Language Model Based Agents: A Survey". In: *arXiv preprint arXiv:2309.07864*. Comprehensive survey of LLM-based agents.

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.