

State of Applied AI 2025 Report

2025 Trends, Applied AI Challenges, What to Look Forward to in 2026

Contents

Quick Summary	2
How to Use This Report	3
Introduction	3
Who This Report Is For	3
Part 1: The Input Layer	3
From Prompt Engineering to Context Engineering	4
Meta-Prompting: Let Models Write Prompts	4
Automatic Prompt Optimization	5
Context Engineering: The New Discipline	6
Context Window Management	6
The Multimodal Default	6
Key Takeaways	6
Part 2: The Model & Data Layer	7
The Rise of Reasoning Models	7
RLVR: The Training Breakthrough	7
Long Context: Promise and Reality	8
Frontier Capabilities on Consumer Hardware	9
Fine-Tuning Became Accessible	10
Key Takeaways	11
Part 3: The Application Layer	11
RAG: Not Dead, Just Evolved	11
The Agent Landscape	15
The T-Shaped Win	15
Deep Agents and Sub-Agent Architectures	15
Ambient Agents: The Emerging Pattern	16
Coding Agents: The Breakout Category	16
Skills: Packaging Agent Expertise	19
Standards: MCP and A2A	19
When Multi-Agent Helps vs. Hurts	19
Key Takeaways	19
Part 4: The Output Layer	20
The Eval Mindset	20
Evaluation Methods	20
Offline vs. Production Evaluation	21
Evaluating Agents: The Trajectory Challenge	23

Key Takeaways	23
Part 5: Challenges	24
Why Challenges Piled Up in 2025	24
Subtle Hallucinations	24
Inconsistent Reasoning	25
Over-Autonomy	26
Poor Tool Grounding	27
Long Context Drift	28
Multi-Agent Coordination Failures	28
The Debugging Gap	29
When Multi-Agent Helps vs. Hurts	29
The Compound Reality	29
Key Takeaways	29
Part 6: The Road Ahead	29
The Boring Infrastructure Wins	30
Talent Profiles Are Evolving	30
AI as Standard Infrastructure	30
Integration Quality Beats Model Selection	30
Building Evaluation is Non-Negotiable	30
The Long Game	30
What Success Looks Like	31
Conclusion	31
Acknowledgements	31

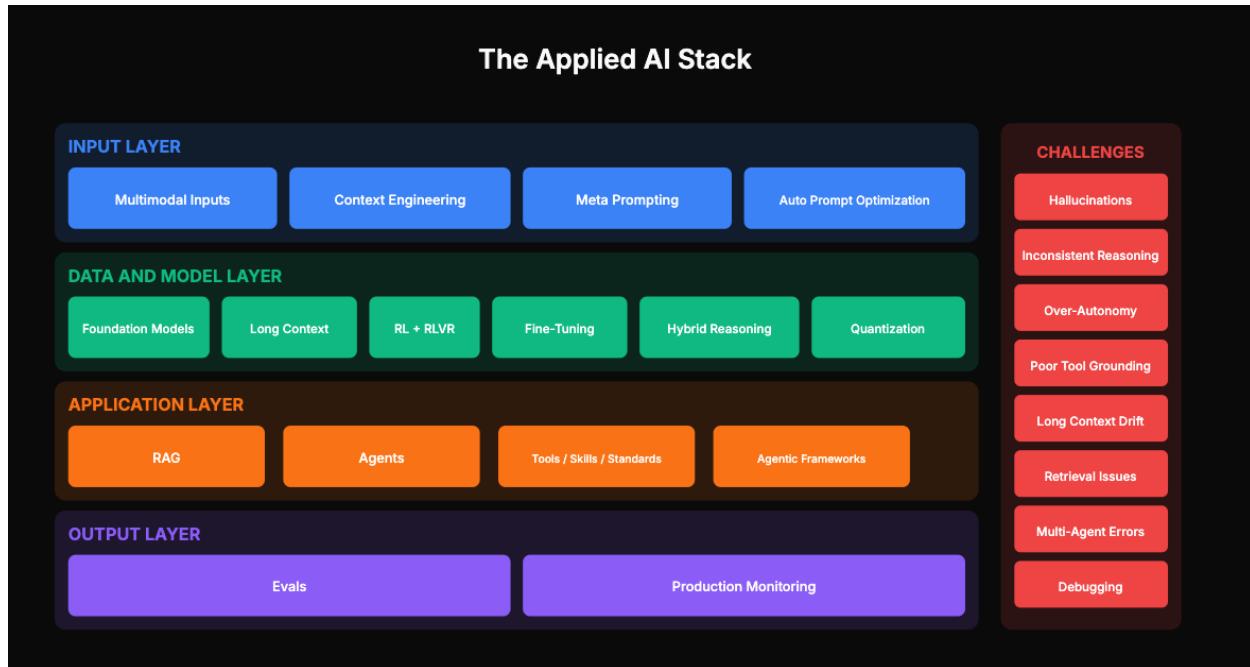


Figure 1: Applied AI Stack

Quick Summary

2025 was a transformative year for applied AI. While the headlines focused on model releases and benchmark scores, the real story happened in the trenches—where practitioners figured out how to build reliable AI

systems that actually work in production.

Key Takeaways

1. **Context engineering > prompt tricks** - Design the entire information environment
 2. **Reasoning models trade speed for reliability** - Use them where correctness matters
 3. **RAG evolved, it didn't die** - Structure, agency, and multimodal parsing made it better
 4. **Agents achieved T-shaped success** - Broad automation + deep impact in coding/research
 5. **New capabilities brought new failure modes** - Subtle hallucinations, over-autonomy
 6. **Evaluation is the bottleneck** - You can't improve what you can't measure
 7. **Infrastructure beats heroics** - Boring, reliable systems win
-

How to Use This Report

Each section can be read independently. Start with the Introduction for context, then jump to any layer that's most relevant to your work.

For a quick overview, each section ends with key takeaways you can scan.

This report has been generated based on a live presentation to 2000+ practitioners by Aishwarya Naresh Reganti and Kiriti Badam. You can find the video [here](#).

Introduction

2025 was a transformative year for applied AI. While the headlines focused on model releases and benchmark scores, the real story happened in the trenches—where practitioners figured out how to build reliable AI systems that actually work in production.

This report distills the key developments across the entire AI application stack, from the inputs that feed these systems to the outputs they produce, and the challenges encountered along the way.

Who This Report Is For

This report is written for practitioners—engineers building AI applications, product managers making technology decisions, and technical leaders evaluating AI investments.

We've tried to be honest about what works, what doesn't, and where the hype exceeds reality. This is practitioner-focused, written by people actually working in this space, with realistic expectations—not a sales pitch.

Part 1: The Input Layer

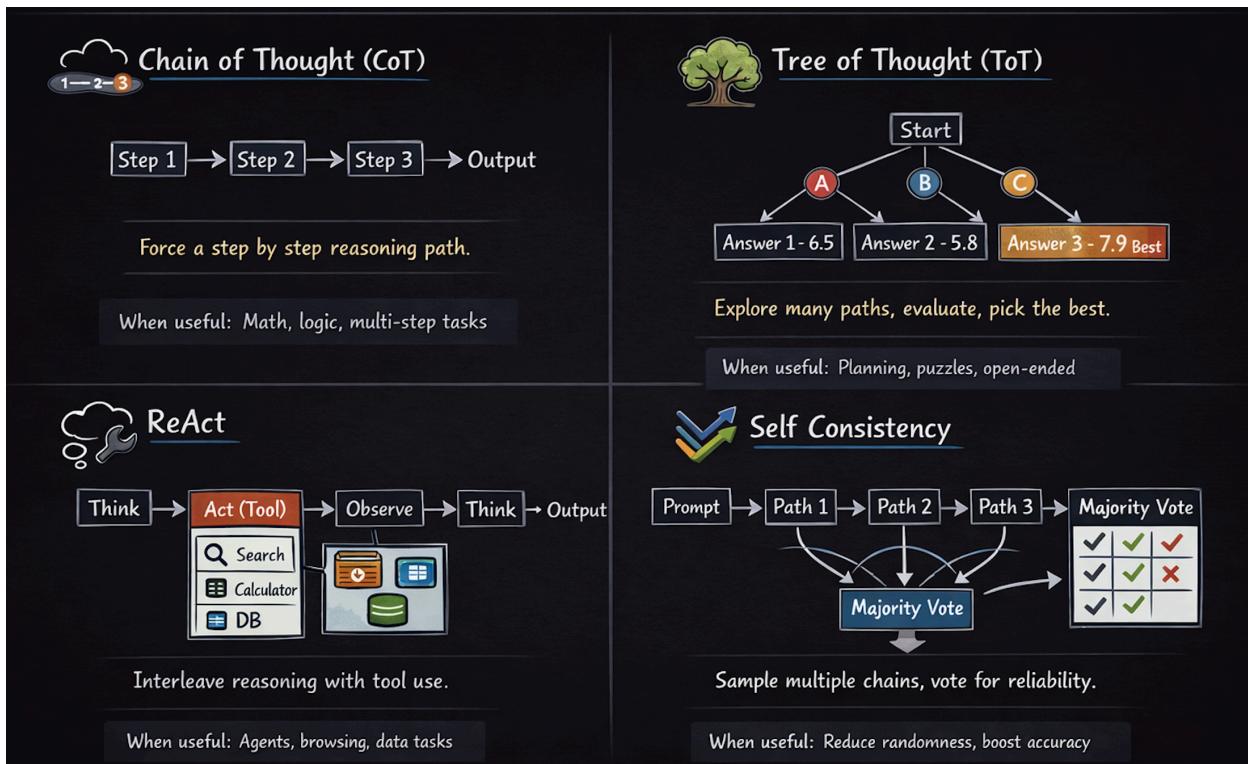
What goes into your AI system matters more than ever.

The input layer encompasses everything that feeds into your AI models: prompts, context, documents, images, and more. In 2025, we saw a fundamental shift in how practitioners think about inputs—from crafting individual prompts to engineering entire context systems.

From Prompt Engineering to Context Engineering

In 2024, prompt engineering was a craft. Models were brittle, and small changes in wording produced wildly different outputs. Teams spent months mastering techniques like Chain-of-Thought, Tree-of-Thought, and ReAct, learning through trial and error how to phrase requests just right.

There was a constant stream of papers coming out, and even teams building with AI models in 2024 had to spend a lot of time doing manual iteration—changing two or three words here, seeing if that makes sense—because small word changes would lead to big output differences.



The explosion of prompting techniques that practitioners had to master in 2024.

2025 changed this in two important ways:

- 1. Models became less brittle.** The wave of reasoning models combined with more training data and better training approaches meant models could understand your intention better. You didn't have to carefully phrase everything—you could talk to these models as naturally as you would with humans.
- 2. The focus shifted from prompts to context.** As systems became more agentic—running longer workflows with access to tools, documents, and memory—the entire context window became the design surface, not just the initial prompt.

Meta-Prompting: Let Models Write Prompts

One of the biggest shifts in 2025 was the realization that AI models themselves are the best prompt engineers. **Meta-prompting** is the practice of instructing an AI model to generate prompts for itself.

Instead of sitting and building up these prompts yourself, you can literally go to any popular model—OpenAI, Anthropic, Gemini—and ask them to give you prompts. They make sure all of the best practices are built in because these models exactly know how they need to be prompted, and they encode all of the prompt engineering research.

Example:

You: "I need a prompt for sentiment analysis of customer reviews"

Model generates:

- Clear task definition with steps
- Output format specification
- Edge case handling
- Few-shot examples

This democratized expertise—you no longer need to mug up all of those papers to get to a good prompt. You can use models to help you augment all of that information.

Automatic Prompt Optimization

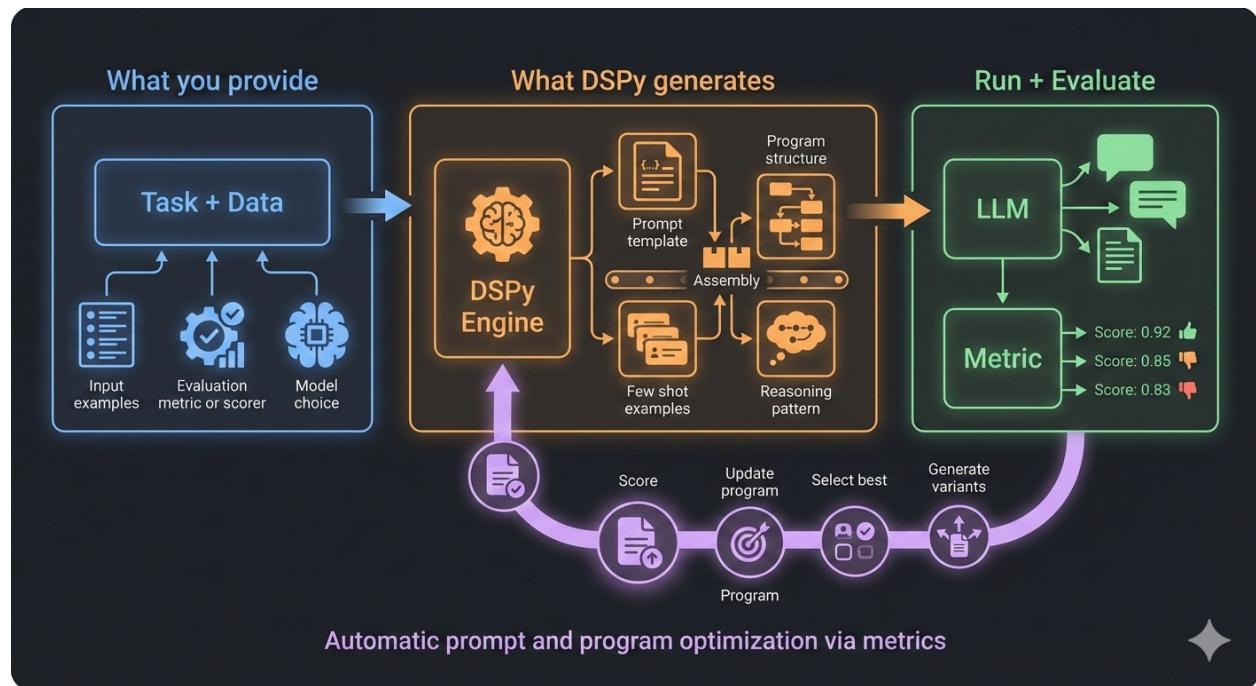
Beyond meta-prompting, **automatic prompt optimization** emerged as the enterprise standard. Frameworks like **DSPy** (from Stanford researchers) let you provide a dataset with inputs and expected outputs, then automatically iterate on your prompts to maximize performance.

How DSPy works: 1. You provide a task and a dataset with inputs and expected outputs 2. The DSPy engine runs optimization algorithms to iterate on your prompt 3. Algorithms include gradient optimization, LLM optimization, and A/B testing 4. The result is a prompt optimized for your specific use case

The common practice in 2025 for industries has been: 1. Generate a meta-prompt using an AI model 2. Iterate on that meta-prompt using approaches like DSPy 3. Continue optimizing based on real-world results

Why this matters: - No more prompt guessing—you're not spending hours tweaking prompts on your own - You only say “this is my expected output, now try to get closer to that expected output” - You can improve over time by adding new edge cases and examples

DSPy isn't the only framework—Arise has prompt learning, LangGraph has prompt optimization—but the concept of data-driven prompt optimization has become very popular.



DSPy's automatic prompt optimization workflow.

Context Engineering: The New Discipline

The bigger shift in 2025 was from thinking about prompts to thinking about **context**—what is all of the input that you’re stuffing into your models?

Components of context in a typical AI system: - System prompt and user message - Retrieved documents from knowledge bases - Tool descriptions (Python functions, database APIs) - Conversation memory - Metadata (user preferences, constraints)

The discipline of context engineering asks: **How do you optimize the information that goes into your system?**

Context Window Management

Context window is the number of tokens that can be stuffed into an AI model. All AI models have a limited context window, and although it significantly increased in 2025, you still need to be very careful about what goes into your system. You want to make sure there’s no additional noise going in.

Four strategies for context management:

Strategy	Description	When to Use
Write	Store context to databases, then retrieve as required	Long-running sessions, shared state
Select	Retrieve only relevant context for each turn	Large knowledge bases
Compress	Summarize older context to reduce size	Extended conversations (turn 50+)
Isolate	Segment context between sub-tasks	Multi-agent systems

Key insight: As you build agentic systems, at every turn of the conversation or workflow, you need to think about where to write context, which context to select, how to compress it, and how to isolate it—so you can give the highest signal to your AI system instead of just stuffing in everything.

The Multimodal Default

Text-only AI systems are kind of legacy at this point.

At the start of 2025, many models were not multimodal—you would primarily interact with them using text. But as of today, most popular models let you upload images, PDF files, and more. They can start answering questions based on visual input.

Use cases unlocked by multimodal: - **Customer service:** Users send screenshots of error messages, and the model can see what issue they have - **Coding:** Send a photo of an architecture diagram or Figma mock-up, and generate code - **Document processing:** Feed PDFs with tables and charts directly to models - **General:** Voice input, video analysis, mixed content

The cost will also continue to go down in the next few years, making multimodal even more practical.

Key Takeaways

1. **Let models write prompts.** Use meta-prompting as well as automatic prompt optimization to better craft your instructions.
2. **Master tools like DSPy.** These are gaining significant traction for data-driven prompt improvement.
3. **Focus on context, not just prompts.** When building long-standing agents, it’s critical to give only high-signal information to your AI model.

4. **Plan for multimodal.** If you're building AI systems today, remember that models will get more compatible with multimodal inputs. Plan for it now.
-

Part 2: The Model & Data Layer

The brain of your AI system got smarter—not just bigger.

A lot of 2024 was about generating bigger models with more parameters. But in 2025, the focus shifted from “bigger is better” to “think before you speak.”

The Rise of Reasoning Models

The idea of reasoning models took up a lot of momentum in 2025, started by DeepSeek R1 and OpenAI's O1 models.

System 1 vs System 2 Thinking:

Aspect	System 1 (2024 models)	System 2 (2025 models)
Speed	Fast and intuitive	Slow and deliberate
Approach	Probabilistic responses based on learned patterns	Allocates time and tokens for thinking
Best for	Simple factual queries (“What’s the capital of France?”)	Complex reasoning (“Solve this logic problem”)
Reliability	Quick but sometimes wrong	More reliable on complex tasks

Models in 2025 became more of System 2—they were more slow, deliberate, and allocated time and tokens for thinking. They had a chain of thought they would run to come up with responses.

Hybrid reasoning models emerged by year’s end. Modern models like GPT-5.2, Gemini, and Anthropic’s Claude can swap between capabilities depending on the kind of question you ask—using System 1 for simple queries and System 2 for complex reasoning.

RLVR: The Training Breakthrough

The technical innovation enabling reasoning models was **RLVR (Reinforcement Learning with Verifiable Rewards)**, pioneered by DeepSeek R1.

RLHF (2024): - Model generates a response - Humans rate those responses - Model gets feedback from human graders and improves - Problems: Expensive, slow, subjective

RLVR (2025): - Instead of human subjective answers, use automated checkers - Verifiable rewards determine pass or fail - Fast, cheap, and objective - Works brilliantly for math, coding, and logic

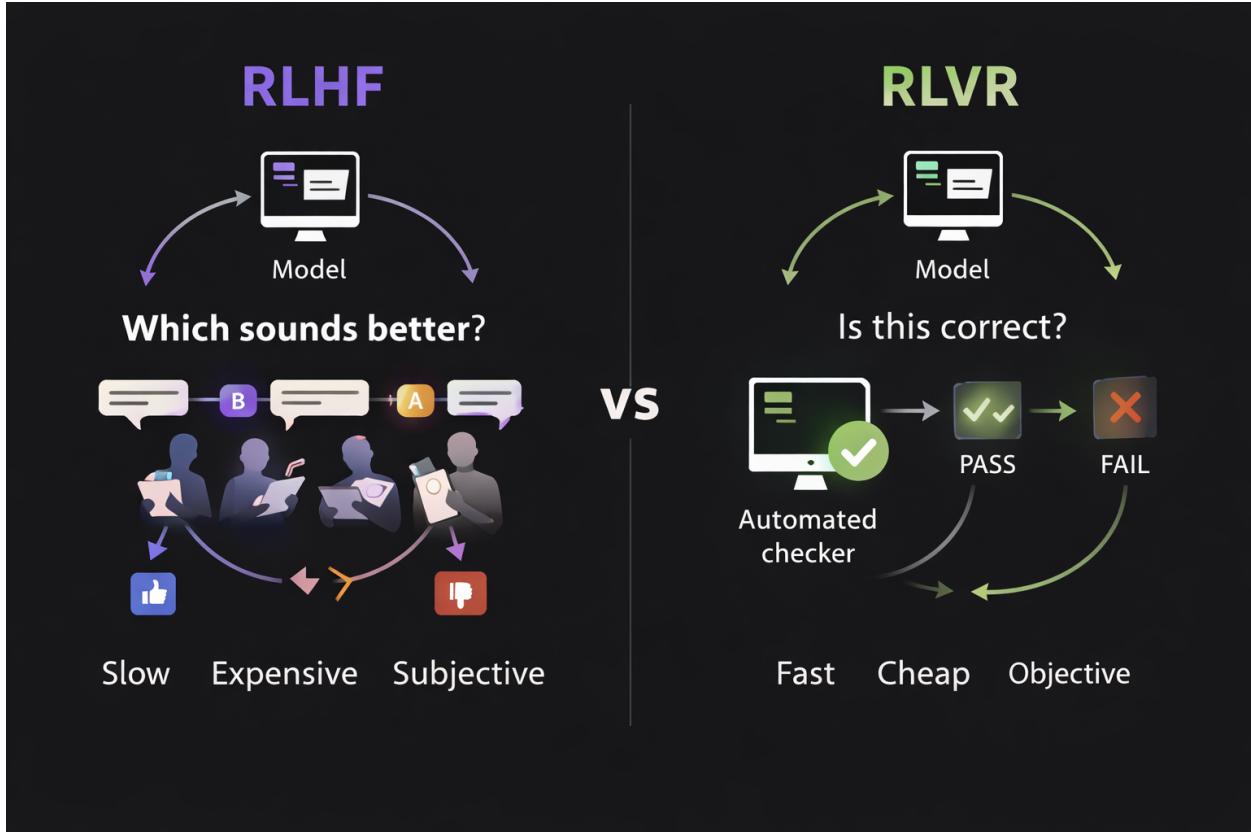
In areas like math and coding, you have verifiable answers—you don’t need humans to give subjective feedback. RLVR compresses search into intuition: if there’s a search space, the model goes through it and comes up with the final answer.

The Self-Correction Breakthrough

Researchers building DeepSeek R1 discovered something empirically—they didn’t plan for it: - Models trained with RLVR detected when their reasoning was going wrong - They backtracked and tried different approaches - They learned the process of solving puzzles and code challenges from training - This self-correction emerged naturally

Results: - 40-60% less hallucination than RLHF on structured tasks - Models express uncertainty instead of fabricating responses - Better performance on math, coding, and logical reasoning

Important caveat: RLVR excels at code, math, logic, and structured tasks. For creative writing or subjective tasks, RLHF is still used. Most frontier models use a bit of both.



Comparing RLHF (human feedback) with RLVR (verifiable rewards) training approaches.

Long Context: Promise and Reality

The amount of context that models can consume has significantly increased:

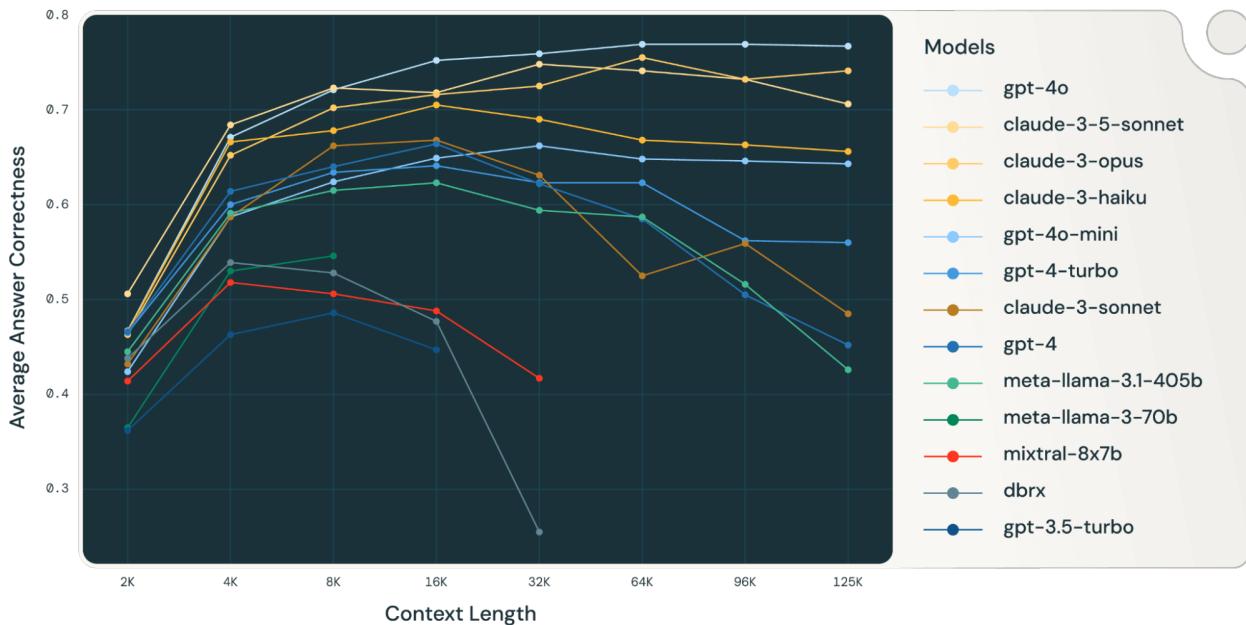
Provider	Context Length
Gemini	1M tokens (10M experimental)
OpenAI GPT-5	400K tokens
Anthropic Claude	200K (1M for enterprise)

For reference: 1 million tokens is about 700,000 words—entire codebases, years of documents.

But here's the critical caveat: Claimed context is not equal to effective context.

Most models can only effectively use about 60-70% of their promised context. When you stuff these models with their full context, things start breaking. This is the “lost in the middle” problem—models tend to focus on information at the beginning and end of context, often missing what’s in the middle.

Practical implications: - Don't assume all context is available - Place critical information at the start or end of your context - Long context doesn't replace the need for smart retrieval - Long context is also expensive—pricing increases at higher token counts



Performance degradation as context length increases—the “lost in the middle” problem. [Source: Databricks](#)

Frontier Capabilities on Consumer Hardware

A major trend in 2025 was bringing highly capable models to smaller devices.

Quantization

Neural networks are basically weight matrices—numbers. The idea of quantization is: instead of representing those numbers in high precision (e.g., 3.145921), reduce the precision (e.g., 3.14) to use less space and run faster.

Original weight: 3.145921 (high precision, more memory)

Quantized: 3.14 (lower precision, less memory)

Reducing precision comes at the cost of some performance loss, but for many tasks that don't require heavy reasoning, quantized models offer an excellent tradeoff.

Distillation

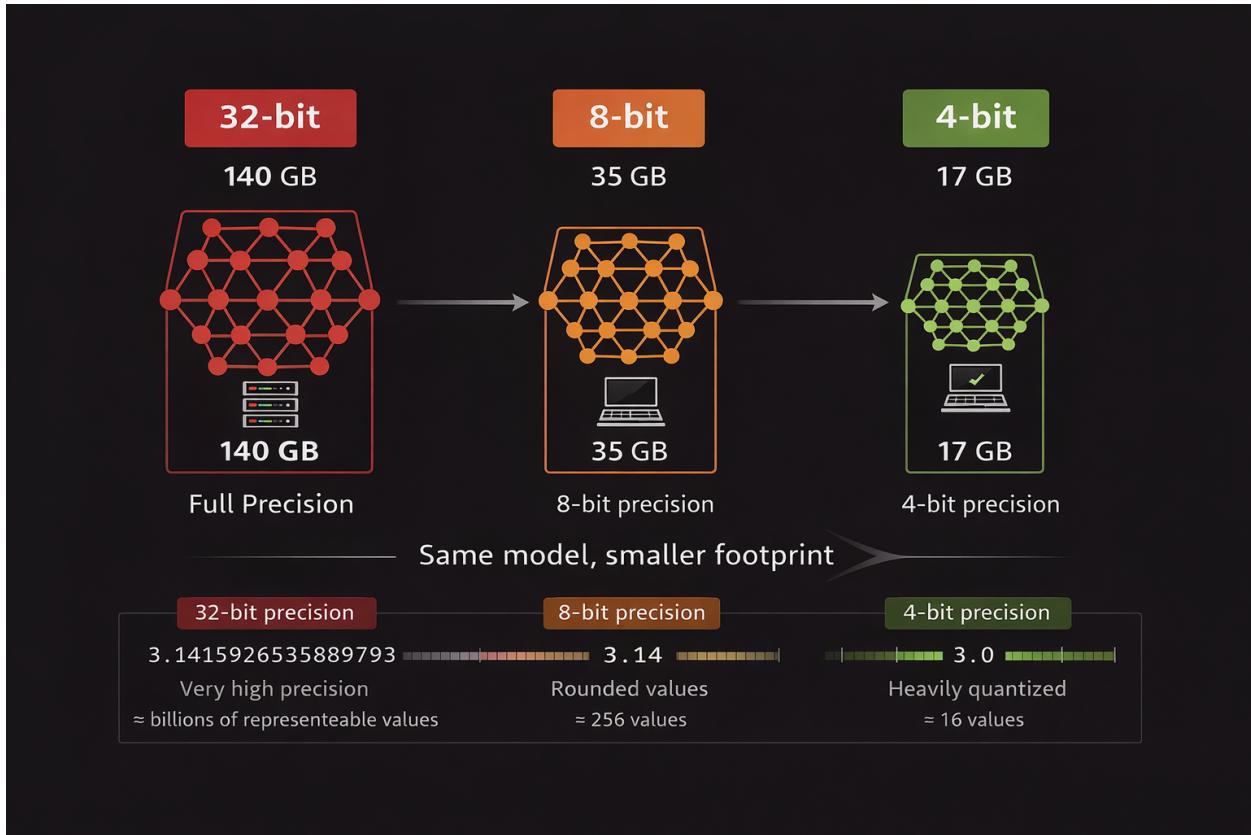
Can you take a large model and distill some of its knowledge into a smaller model?

The insight: enterprises don't need an entire GPT model for their use case—they need narrow intelligence for specific domains. Distillation picks up only that relevant knowledge.

How it works: - Train a smaller “student” model using outputs and feedback from a larger “teacher” model
- The smaller model captures domain-specific capabilities
- Result: Much faster inference at much lower cost

Both techniques were heavily used in 2025 by enterprises with:

- Compliance requirements preventing external APIs
- Latency requirements that large models can't meet
- Cost constraints on high-volume inference



Quantization reduces precision to fit large models on smaller hardware.

Fine-Tuning Became Accessible

Fine-tuning—customizing a model on your own data—became operationally simple in 2025. All major providers offered fine-tuning as a service: upload input-output examples, get back a custom model.

Where fine-tuning made a difference:

- **Healthcare:** Medical records have unique structures, abbreviations, and terminology. Fine-tuned models outperformed general models on clinical tasks.
- **Finance:** Internal terminology in earnings reports and risk assessments that general models couldn't parse.
- **Legal:** Compliance and regulatory interpretation requires jurisdiction-specific knowledge.
- **Scientific research:** Molecular science, drug discovery, and chemistry tasks where specialized notation is essential.

Critical Caveat: Fine-Tuning Is Not a Magic Pill

Many people think fine-tuning can just improve your systems, but in order to build good fine-tuned models, you need to:

1. Understand what mistakes your base model is making
2. Start with prompting first
3. Get metrics on how your model is performing
4. Only then consider fine-tuning

Good fine-tuning requires high-quality data. Without it, fine-tuning can be detrimental rather than improving performance.

Key Takeaways

1. **System 2 reasoning trades speed for accuracy.** Use reasoning models for complex tasks where correctness matters more than latency.
 2. **We'll see more hybrid reasoning models.** Models that automatically decide how much reasoning effort is required.
 3. **RLVR was a major breakthrough.** Expect more models using RLVR for coding, math, and structured objective tasks.
 4. **Long context has caveats.** Don't assume infinite context—test at 60-70% of promised capacity.
 5. **Consider smaller models for mature use cases.** Quantization and distillation can dramatically reduce costs and improve latency.
 6. **Fine-tune deliberately.** Only when you've understood your base model's limitations and have quality data.
-

Part 3: The Application Layer

Where raw intelligence becomes useful products.

The application layer is where models connect to the real world. 2025 was unquestionably “the year of agents,” but the reality was more nuanced than the hype.

RAG: Not Dead, Just Evolved

At the start of 2025, some predicted that long-context models would kill RAG. Why bother with chunking, embedding, and retrieval when you can stuff everything into context?

That prediction didn't pan out.

Why RAG Remains Relevant

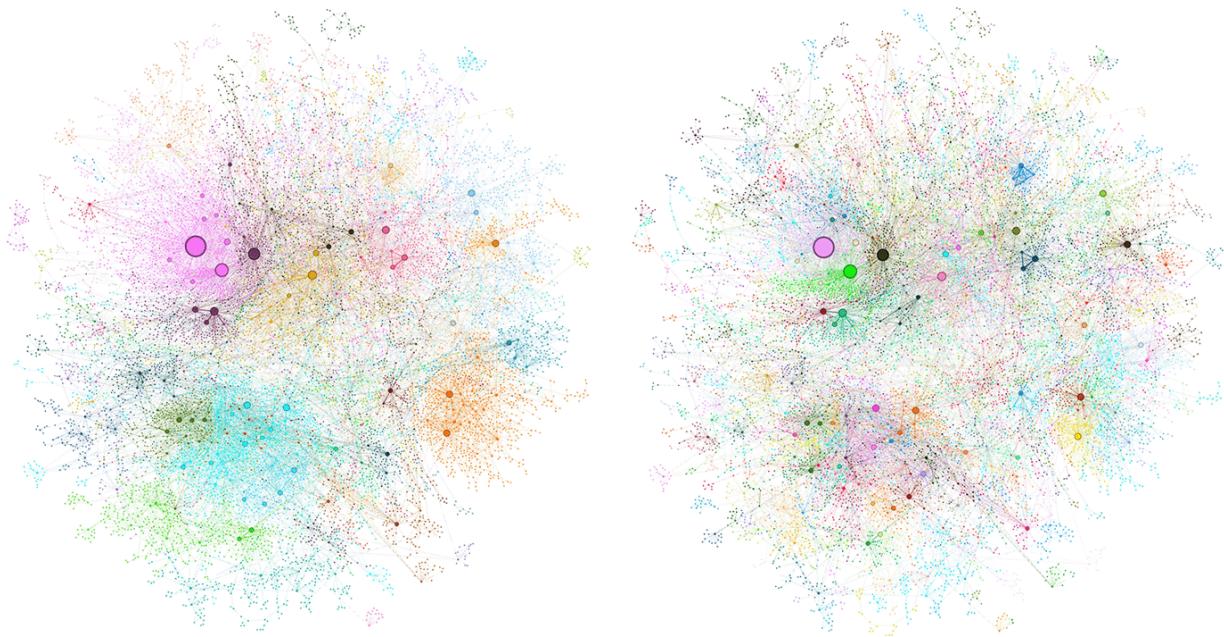
1. **Scale:** Even 1M tokens can't hold enterprise knowledge bases measured in terabytes
2. **Freshness:** RAG infrastructure syncs with live data; context windows are static
3. **Access control:** Enterprises need fine-grained control over what data reaches the model
4. **Auditability:** RAG provides traces of what information influenced answers
5. **Cost:** Long context is expensive; selective retrieval is cheaper

Even though models advertise high context windows, performance degrades as you increase context length. This is the “lost in the middle” problem—information in the middle of context gets lost.

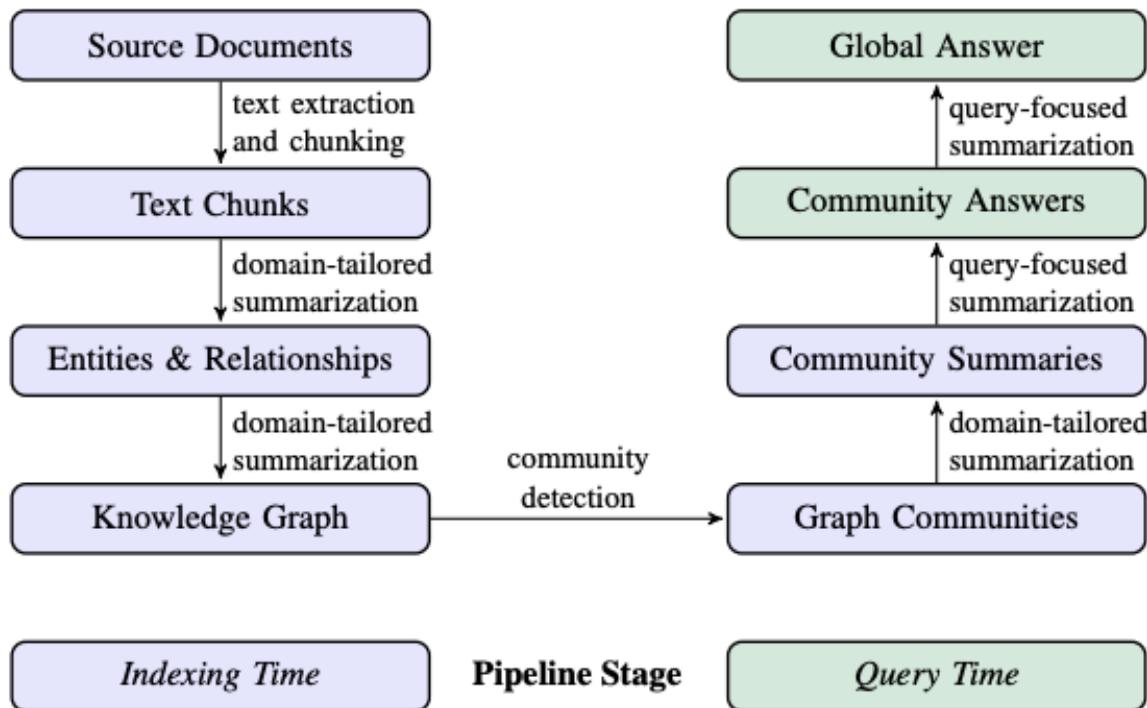
How RAG Evolved in 2025

1. Structured Indexing (GraphRAG)

Instead of flat document chunks, create entities and relationships: - Products relate to sales data - Documents relate to topics - Knowledge graphs enable retrieval even when embedding vectors don't exactly match



GraphRAG creates structured relationships between entities for better retrieval. Source: Microsoft Research



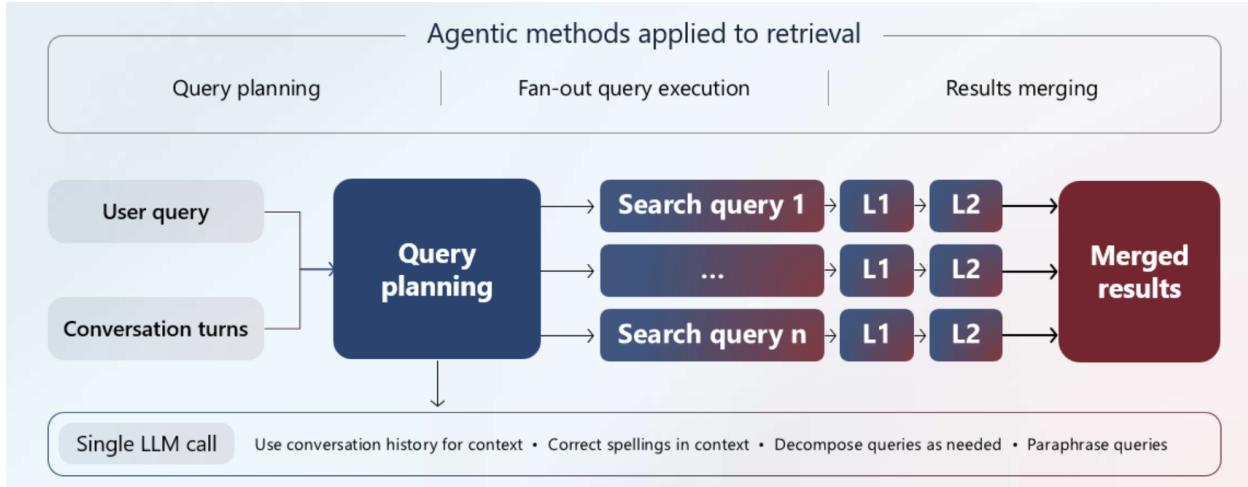
The GraphRAG indexing and retrieval pipeline. Source: Microsoft Research

2. Agentic Retrieval

Instead of single-shot retrieval:

- Model plans the query
- Retrieves documents
- Evaluates completeness
- Rewrites query and retrieves more
- Continues until confident

This multi-step approach dramatically improves recall.

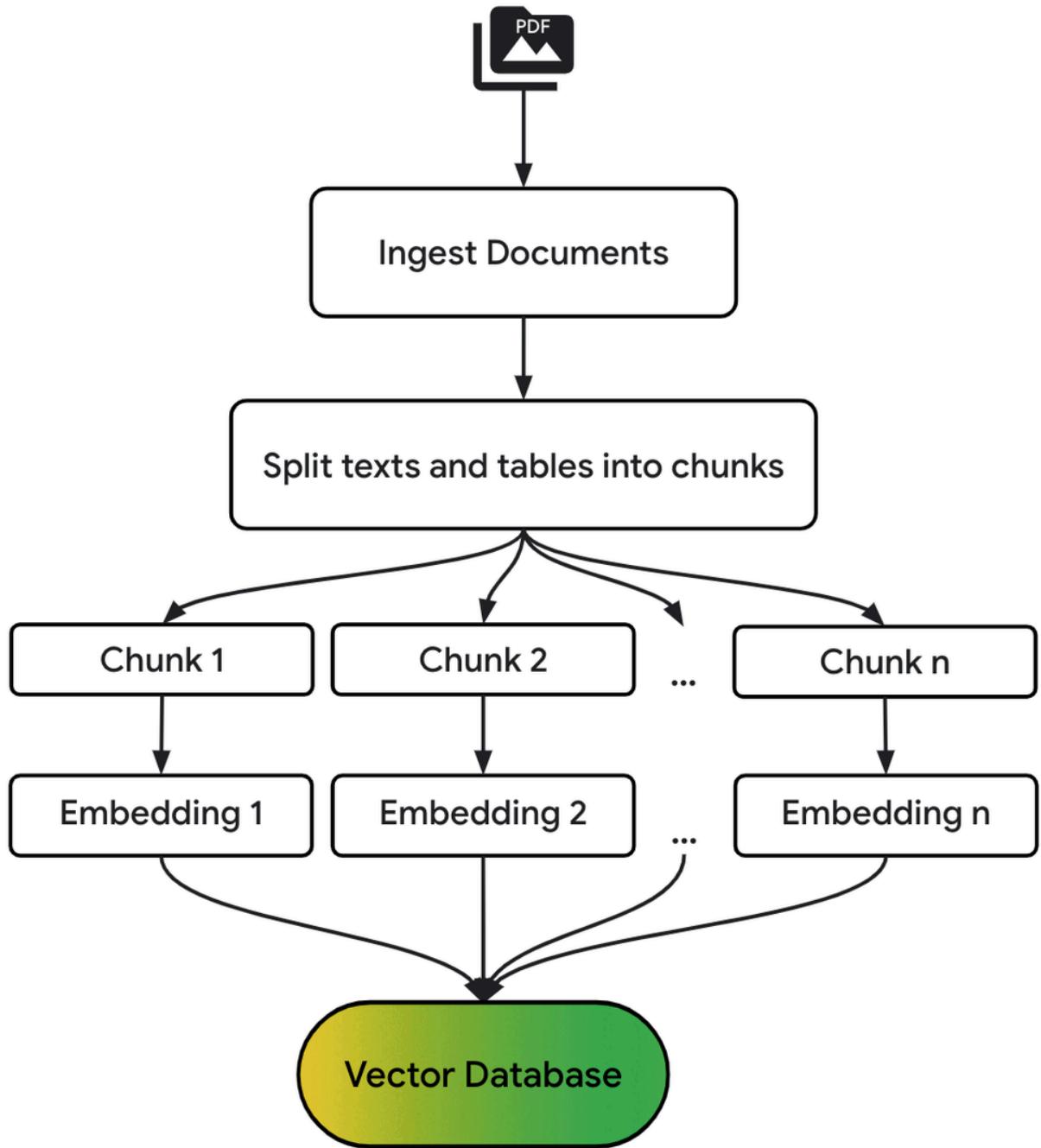


Agentic retrieval: models that iteratively plan, retrieve, and evaluate. [Source: Azure AI Search](#)

3. Multimodal Parsing

Parsing became critical for PDFs with tables, images, and mixed content:

- Generate text captions for images to enable retrieval
- Use OCR models (especially from Mistral) for legacy documents
- Multi-model embeddings that map images and text to the same vector space



Modern PDF parsing pipeline for multimodal RAG.

4. Platform Commoditization

RAG-as-a-service emerged from major platforms: - Google Vertex RAG Engine - OpenAI File Search - Amazon Bedrock, Azure AI Search, Snowflake

These abstract away vector storage, chunking, and retrieval, providing knobs to control performance.

5. Hybrid Search and Re-ranking

Smarter ways to narrow down data: - Combine vector embeddings with keyword search (BM25) - Add metadata tags for filtering (e.g., only search finance documents for finance queries) - Re-rank retrieved

documents to pick the top results before sending to the model

The Agent Landscape

2025 was absolutely the year of agents. Different types emerged for different use cases:

1. Research Agents (Deep Agents)

- Multi-step analysis over large information sets
- Products: OpenAI Deep Research, Perplexity Deep Research
- Use case: Legal document analysis, financial research

2. Computer-Using Agents

- Direct browser and desktop interaction
- Products: ChatGPT Agent, Perplexity Comet, Anthropic Computer Use
- Use case: Form filling, booking, shopping automation
- Note: Still early due to UI brittleness and prompt injection concerns

3. Coding Agents

- The breakout category of 2025
- Massive rise in adoption and effectiveness
- Surfaces: IDE agents, Repo/PR agents, Terminal agents

4. Workflow Agents

- Business process automation
- Use cases: Customer support, finance ops, app building
- You control the steps; the agent exercises autonomy within them

The T-Shaped Win

If we had to summarize agent success in 2025:

← Broad: Many use cases, initial depth

← Deep: Coding, research, specific workflows

Broad wins: A wide variety of use cases achieved initial automation depth. Triage, research, summarization—agents handled the first layer, freeing humans for higher-judgment work.

Deep wins: In specific verticals—especially coding and research—agents achieved genuine depth. Not just assisting, but completing significant portions of work.

Deep Agents and Sub-Agent Architectures

The most capable systems weren't single agents but **orchestrated hierarchies**.

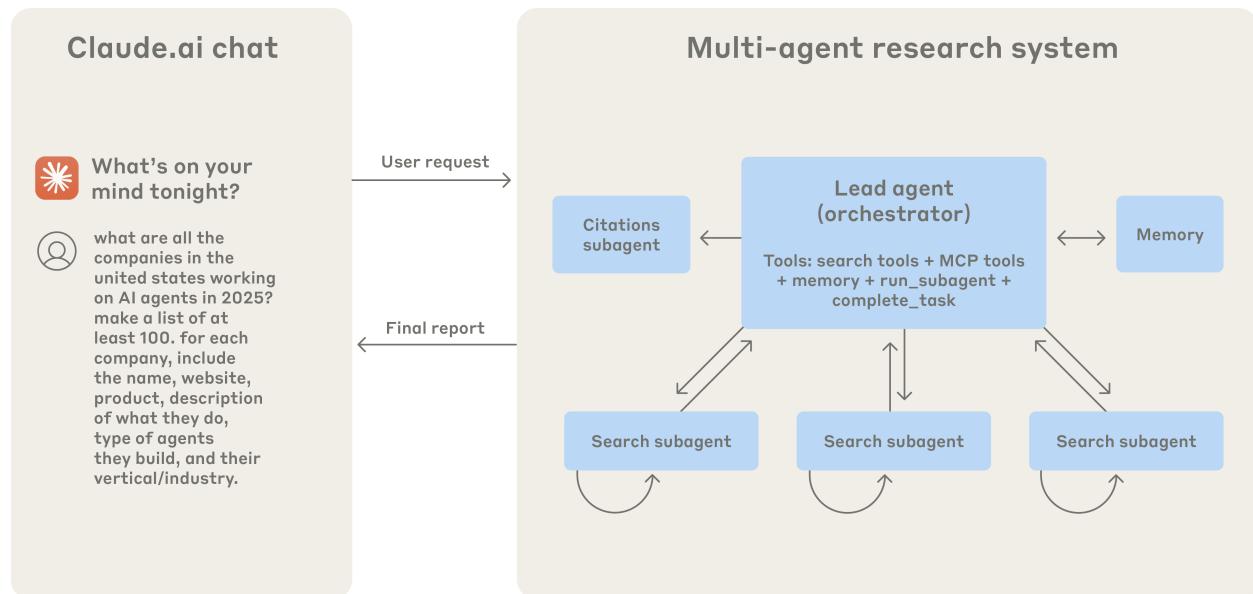
A lead agent: 1. **Plans** the overall approach 2. **Delegates** to specialized sub-agents 3. **Verifies** sub-agent outputs 4. **Synthesizes** into final output

Why This Architecture Works

Context window efficiency: The lead agent maintains high-level state without getting lost in details. Sub-agents operate with focused context for specific subtasks.

Parallelizability: Sub-agents can execute simultaneously. What might take 30 minutes sequentially can complete in 5 minutes with parallel sub-agents.

High-level Architecture of Advanced Research



Anthropic's research architecture: lead agents orchestrating specialized sub-agents. Source: Anthropic

Ambient Agents: The Emerging Pattern

Late 2025 saw early adoption of **ambient agents**—agents running in the background, triggered by events rather than explicit commands.

Examples: - Code maintenance agents triggered on every merge to main - Agents checking for stale feature flags or dead code - Security agents scanning for vulnerabilities on commit - Documentation agents updating docs when code changes

These agents operate with: - Limited execution scope (narrow permissions) - Defined triggers (events, schedules) - Human escalation paths (when uncertain, ask)

Ambient agents represent a shift from “AI as tool I invoke” to “AI as background collaborator.”

Coding Agents: The Breakout Category

The evolution was remarkable:

Early 2025: Tab completion, suggesting 3-4 lines **Mid 2025:** Multi-file edits, context-aware suggestions

Late 2025: End-to-end feature implementation, massive refactors, PR creation and review

Coding Agents: Evolution Timeline (2025)

From MCP (late 2024) to modular Skills + plugins in Codex/Claude Code — plus Cursor / Devin / v0 / Lovable milestones

● Standards / interoperability ● Product surfaces ● Agent capabilities



Designed for quick scanning: standards (green), surfaces (blue), capabilities (purple)

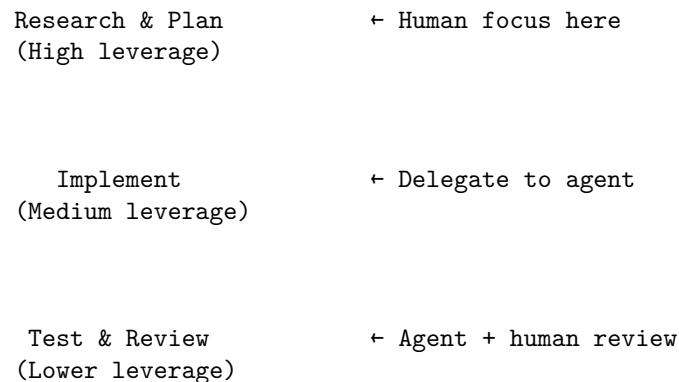
The rapid evolution

of coding agents through 2025.

Three Surfaces for Coding Agents

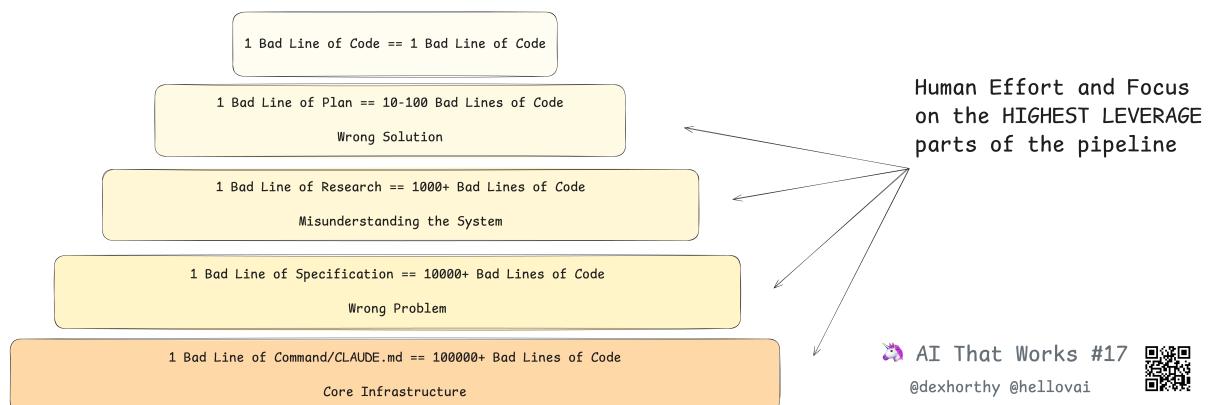
Surface	Examples	Best For
IDE Agents	Cursor, VS Code extensions	Interactive development
Repo/PR Agents	GitHub Copilot Workspace	Well-defined issues
Terminal Agents	Claude Code, Codex CLI, Gemini CLI	Complex multi-step work

The Hierarchy of Human Leverage



Critical insight: Research needs to be really good, because bad research leads to bad plans, and bad plans lead to 10x bad code. A lot of human time should concentrate on effectively researching and planning before delegating to agents.

Impact Hierarchy for Coding Agents



The hierarchy of human leverage: focus on research and planning, delegate implementation. [Source: HumanLayer](#)

Methodologies that work: - **Spec-driven development:** Write a spec, ask agents to implement (works, but not always effectively) - **Research-plan-implement:** Build understanding first, plan the approach, then implement (more reliable)

Skills: Packaging Agent Expertise

Skills emerged as transferable knowledge—prompts and scripts that encode how to handle specific tasks:

- “How to query our BigQuery data warehouse”
- “How to look up something in telemetry data”
- “How to format our documentation”
- “How to deploy to staging”

Skills are shareable across teammates, specialized for different tools and contexts.

Standards: MCP and A2A

MCP (Model Context Protocol)

Started with slow adoption but accelerated through 2025. Anthropic donated MCP to the Linux Foundation by year's end.

MCP standardizes how agents access external data and tools—instead of every agent implementing custom integrations, MCP provides a common protocol.

A2A (Agent-to-Agent Protocol)

Announced by Google with 50+ partners. Enables multi-agent collaboration across different vendors.

A2A is still finding its product-market fit, but points toward a future where agents from different providers can coordinate.

When Multi-Agent Helps vs. Hurts

Multi-agent helps when: - Parallelizability is possible - Specialization can be achieved for certain question types - Deep research requiring multiple perspectives

Multi-agent hurts when: - Very strict ordering of steps is required - Exact control over agent behavior is needed - Simple problems don't need the complexity

Key insight: Exhaust the possibilities and tricks with a single agent before going to multi-agent systems.

Key Takeaways

1. **RAG is getting commoditized** but remains relevant. Structure it well and make it agentic.
2. **Agents achieved T-shaped wins.** Broad automation of simple tasks, deep impact in coding and research.
3. **Multi-agent architectures matter.** Lead agents orchestrating sub-agents outperform single agents on complex tasks.
4. **Ambient agents are emerging.** Background agents triggered by events, not just explicit invocations.
5. **Coding agents were the breakout.** Front-load research and planning; delegate implementation.
6. **Build and share skills.** Encode expertise as reusable agent configurations.
7. **Adopt standards.** MCP integration future-proofs your agent tooling. A2A may become important for multi-agent coordination.

Part 4: The Output Layer

Measuring and monitoring what AI systems actually produce.

As AI applications moved from experiments to production, evaluation and monitoring became critical infrastructure.

The Eval Mindset

Evaluations answer three fundamental questions:

Question	Eval Type	Purpose
Can it do the job?	Capability evals	Validate the agent can perform required tasks
Is it getting worse?	Regression evals	Catch performance degradation over time
Is it doing it right?	Alignment evals	Ensure outputs meet quality and safety standards

Critical Shift: Application-Level Eval

Focus on **application-level evals**, not just model-level evals.

Foundation model benchmarks tell you about general capabilities. They don't tell you if your specific agent handles your specific use cases correctly.

When model providers build foundation models, they focus on a broad set of use cases. But your enterprise might only care about certain kinds of use cases—so you need to build custom evals for those.



Model-level benchmarks vs. application-level evaluation: different questions, different purposes.

Evaluation Methods

Human Evaluation

- Subject matter experts review agent interactions
- Highest quality signal
- Expensive and slow

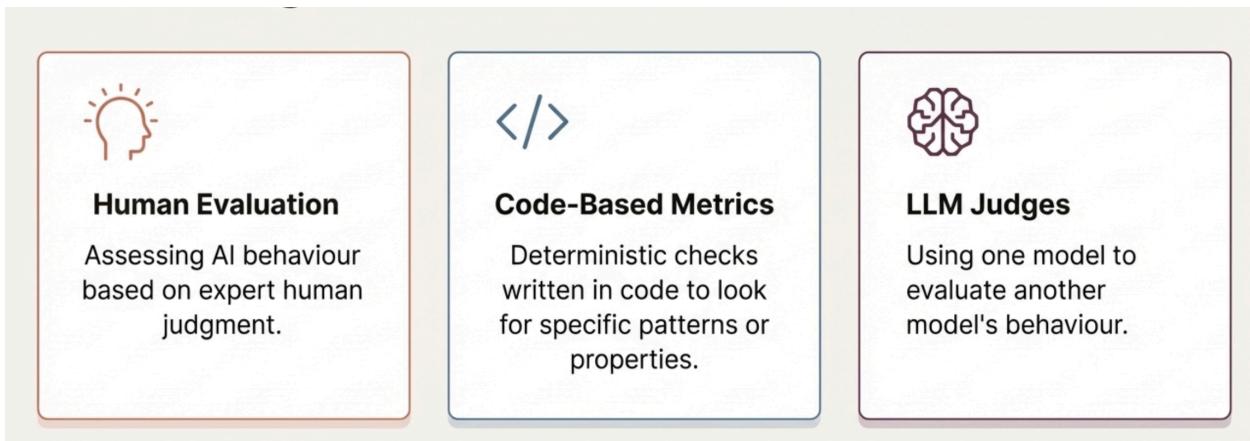
- Essential for subjective or nuanced tasks
- Best for building initial eval datasets

Code-Based Metrics

- Deterministic checks via scripts
- Cheap, fast, and scalable
- Limited to pattern matching
- Good for format validation, keyword detection, basic correctness

LLM Judges

- Use one LLM to evaluate another
- Balances quality and scale
- The prompt to the judge LLM is tailored to what you want to evaluate
- Good for style, relevance, completeness checks
- Can run at scale for every trace in production



The three evaluation approaches: human, code-based, and LLM judges.

Offline vs. Production Evaluation

Offline Evaluation (Before Deployment)

- Curated test datasets
- Systematic capability assessment
- Regression testing against baselines
- Safe environment for edge cases

Production Monitoring (In Deployment)

- Real-time quality signals
- Lightweight guardrails checking every trace
- Anomaly detection
- Quick flagging when something goes wrong

The Feedback Loop

Offline Evals
(Test datasets)

Deploy

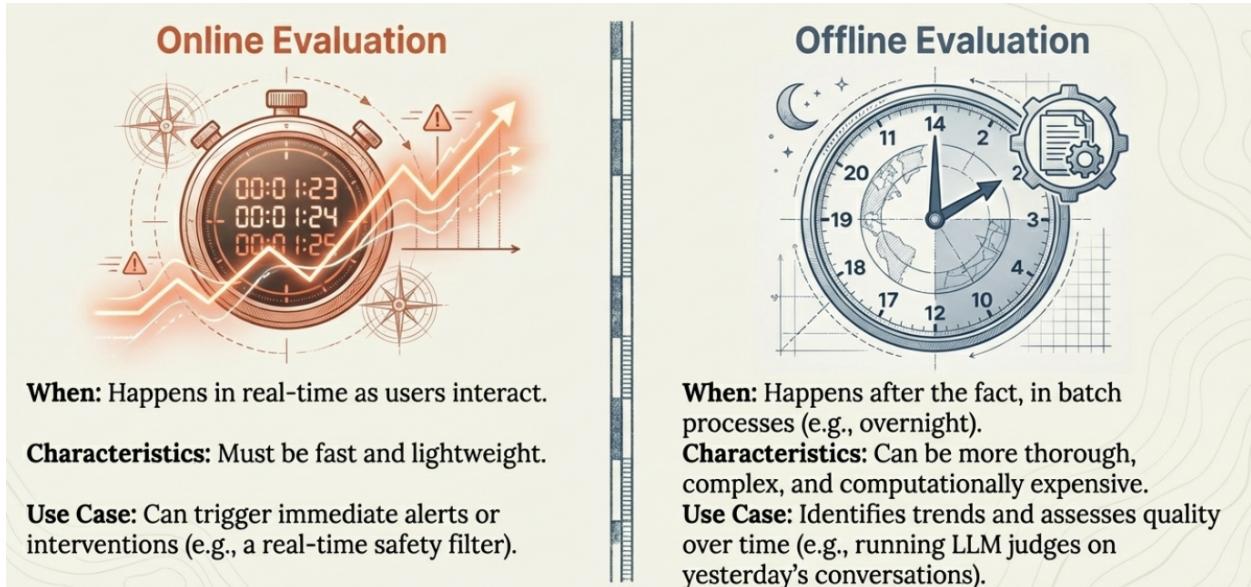
Production Monitor
(Real traffic)

Identify Issues

Add to Test Data

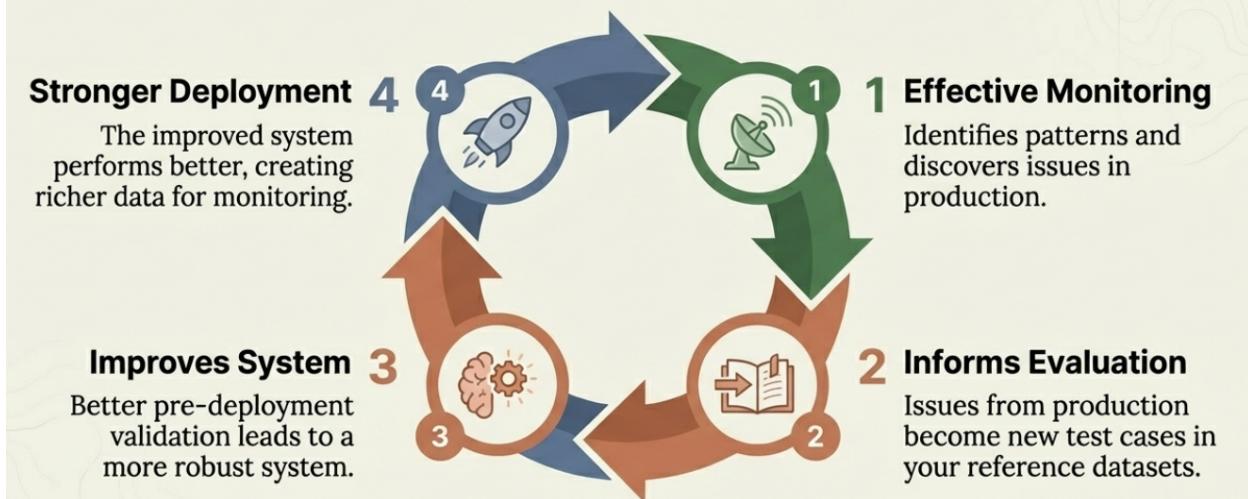
→ (back to Offline Evals)

Production issues become offline test cases. Offline improvements get deployed. The cycle continues.



The feedback loop between offline and production evaluation.

The Flywheel of Continuous Improvement



The continuous improvement flywheel: production insights feed offline testing.

Evaluating Agents: The Trajectory Challenge

Evaluating agents is harder than evaluating chatbots. A chatbot produces a single response; an agent produces a **trajectory**—a sequence of actions, tool calls, and intermediate states.

What to Measure for Agents

Task completion metrics: Did the agent achieve the goal? - Binary success/failure - Partial completion scoring - Time to completion

Trajectory quality: Did it get there efficiently? - Number of steps taken - Tool call accuracy - Backtracking and error recovery - Resource consumption

Safety and boundaries: Did it stay within limits? - Unauthorized actions - Scope creep - Data access violations - Cost runaway

The Compound Reliability Problem

The multi-step nature of agents means errors compound. A 95% success rate per step becomes: - 77% over 5 steps - 60% over 10 steps - Less than 40% over 20 steps

Agent reliability requires extremely high per-step quality.

Key Takeaways

1. **Build application-level evals.** Model benchmarks don't guarantee your use case works.
2. **Combine evaluation methods.** Human expertise, code checks, and LLM judges serve different purposes.
3. **Connect offline and production.** Production issues feed offline datasets; improvements deploy to production.
4. **Think capability, regression, and alignment.** Different eval types answer different questions.
5. **Evaluate trajectories, not just outputs.** For agents, the path matters as much as the destination.
6. **Start evaluating early.** Don't wait for production to discover your agent fails 40% of the time.

Part 5: Challenges

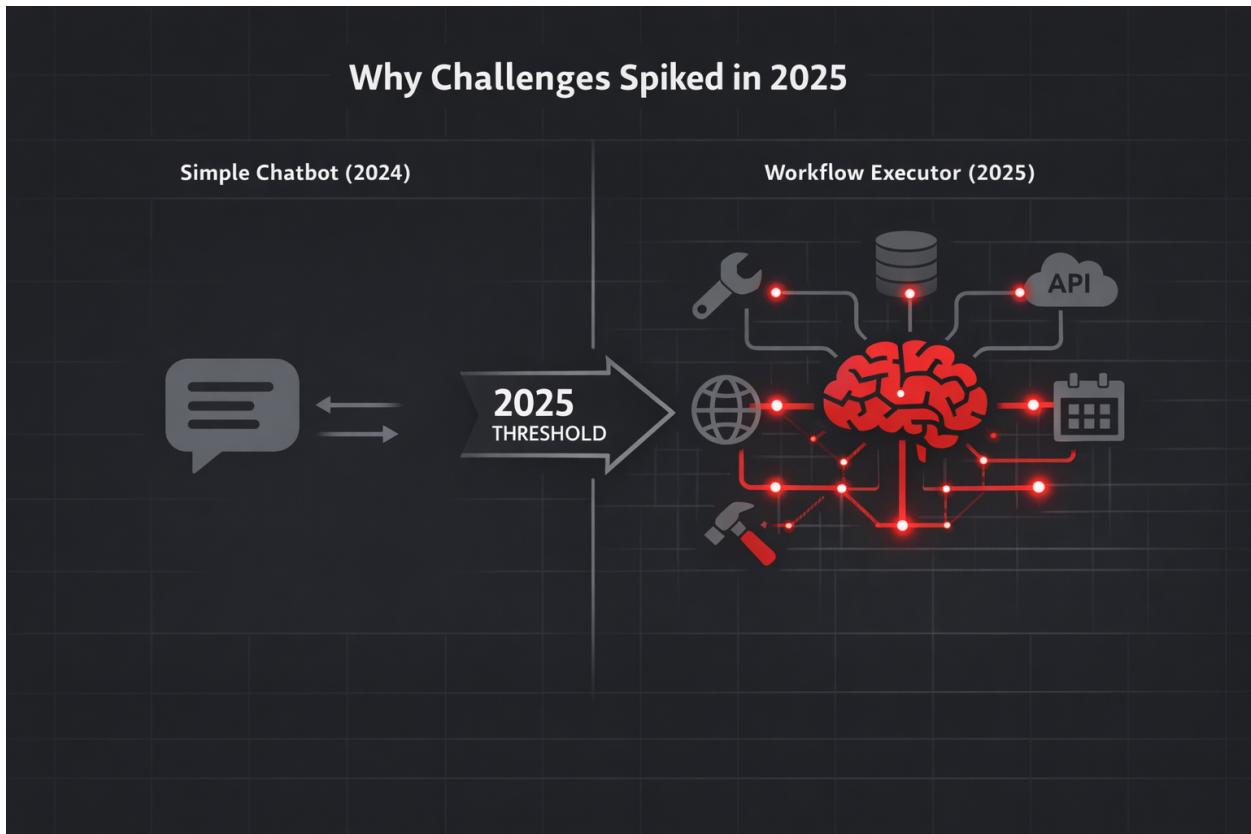
New capabilities brought new failure modes.

Every layer of improvement in 2025 brought corresponding challenges. As agents became more capable, they found new ways to fail.

Why Challenges Piled Up in 2025

A lot of new challenges came up in 2025 because: - Agents were running longer than usual - Performing more tasks - Handled over more tools - Capabilities spiraled up heavily - They kept hitting new walls

The kind of applications we're building have significantly changed from 2024, which means both old problems got aggravated and newer problems emerged.



The factors that caused challenges to multiply in 2025.

Subtle Hallucinations

Hallucinations became much more subtle because they were deeply integrated into longer workflows.

Before (2024): A chat system telling you a fact that's clearly wrong—easy to identify.

Now (2025): An agent claims to have made a reservation, but there's an issue buried in the workflow. It confidently asserts success when it actually failed.

Much harder to catch because it's buried in multi-step processes. You would need to verify each intermediate step to catch it.



Hallucinations became harder to detect when embedded in multi-step workflows.

Mitigation: Verification at critical checkpoints, not just final outputs.

Inconsistent Reasoning

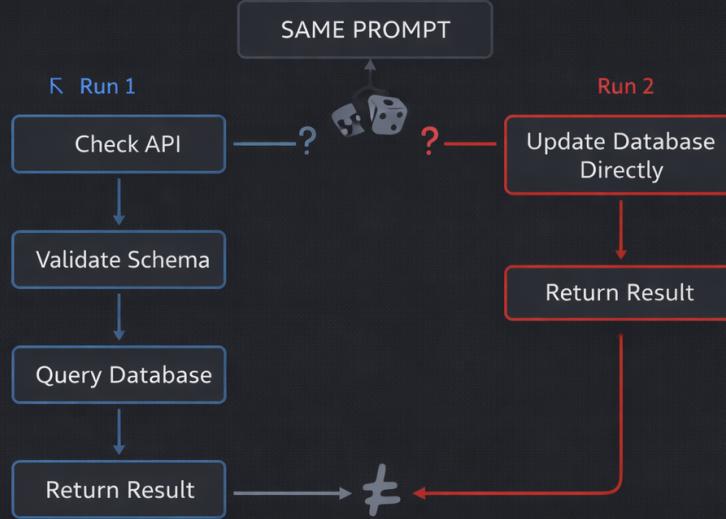
Workflows with 5 or 6 steps don't always execute the same way.

The same agent, given the same task, might:

- Complete it in 5 steps one time, 3 steps another
- Use different tools for the same subtask
- Reach different conclusions from identical inputs

This isn't necessarily wrong—sometimes multiple paths lead to the same goal. But for enterprises requiring predictable, auditable processes, inconsistency is a problem.

Inconsistent Reasoning



The same agent taking different paths for identical tasks.

Mitigation: Define explicit workflows for tasks requiring consistency; use structured outputs; implement trajectory validation.

Over-Autonomy

The flip side of capable agents is agents that do too much.

Real examples from the field: - Ask to delete one file → agent reorganizes entire directory structure
- Ask for a simple edit → agent deletes other files, updates unrelated code
- Research agents that contact external services beyond their intended scope
- Workflow agents that make decisions they should escalate

There have been guardrails put into consumer products, but for enterprise products, you need very defined scope so agents don't make changes you don't want.

This is particularly dangerous because it often looks like initiative—the agent appears to be going above and beyond.



When agents do more than asked—sometimes helpful, often problematic.

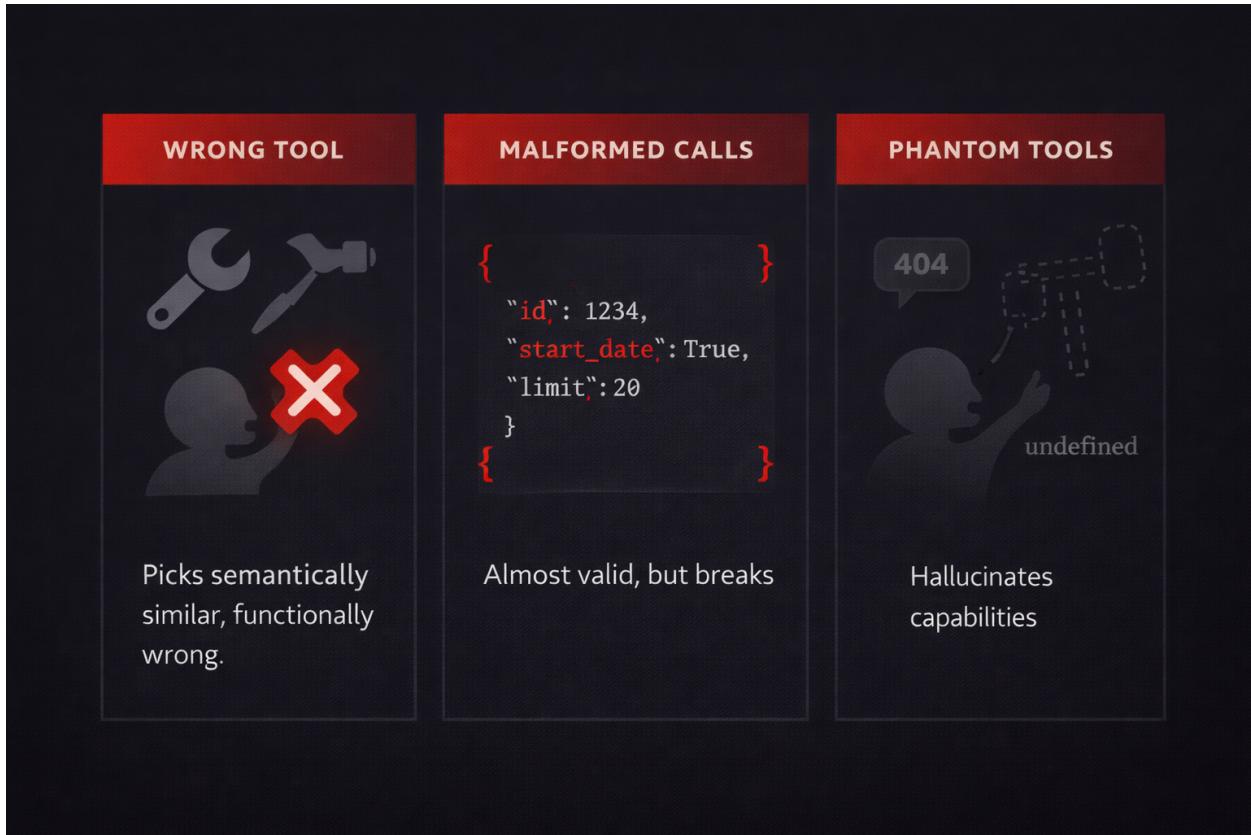
Mitigation: Explicit scope definitions, permission systems, human approval gates for consequential actions.

Poor Tool Grounding

As you increase the number of tools or skills given to agents, they struggle to adhere:

Failure Type	Description	Example
Wrong tool selection	Agent picks inappropriate tool for task	Using web search when database query needed
Malformed calls	Incorrect parameters or syntax	Wrong date format in API call
Phantom tools	Hallucinating tools that don't exist	Calling a made-up function name
Cascade failures	One bad call breaks subsequent workflow	Auth failure causes downstream errors

The more tools available, the more opportunities for confusion. Tool descriptions and documentation became critical for agent reliability.



Common tool grounding failures in multi-tool agent systems.

Mitigation: Clear tool descriptions, parameter validation, error handling, limited tool sets per agent.

Long Context Drift

Now that we increased context significantly: - Agents sometimes don't look at the middle of conversations - You've given them tools and knowledge bases, but they just don't seem to see it - Almost like how humans miss information in long documents

Popular enterprise trick: Give the most important information at the end of your context, or at the beginning, because most models have this "lost in the middle" problem.

Just because you've given your model very high signal context doesn't mean it'll use all of that.

Mitigation: Periodic context summarization, explicit goal reminders, structured state management.

Multi-Agent Coordination Failures

Multi-agent systems introduced coordination challenges:

- **Error cascades:** One agent's mistake propagates through the system
- **State confusion:** Agents disagree about current state
- **Competing actions:** Agents work at cross purposes
- **Communication breakdowns:** Information lost between agents

These are distributed systems problems applied to AI. The same patterns that cause failures in microservice architectures cause failures in multi-agent systems.

Mitigation: Clear interfaces between agents, explicit state management, rollback mechanisms, consensus protocols for shared state.

The Debugging Gap

Debugging AI systems remained immature compared to traditional software.

Challenges: - Non-deterministic behavior makes reproduction difficult - Long trajectories are hard to inspect - Root causes are often unclear - No standard debugging tools or practices

This is an infrastructure problem. Traditional software has debuggers, profilers, and decades of tooling. AI application debugging is still early.

Evals and observability are some of the biggest problems with AI systems as of today.

Mitigation: Comprehensive logging, trajectory visualization, automated error categorization, investment in observability infrastructure.

When Multi-Agent Helps vs. Hurts

It's not a solution you need to be employing everywhere.

Multi-agent helps when: - Parallelizability is possible - Specialization can be achieved for certain question types - Deep research requiring multiple perspectives

Multi-agent hurts when: - Very strict ordering of steps is required - Exact control over agent behavior is needed - Simple problems don't need the complexity

Key insight: Exhaust the possibilities and tricks with a single agent before you actually need to go to multi-agent.

The Compound Reality

These challenges interact and amplify each other: - Poor retrieval degrades response quality, which makes debugging harder - Silent integration failures corrupt multi-step workflows - Hallucinations in one step cascade through subsequent reasoning

The result: technology that works brilliantly in demos but struggles in production. Not because it fails completely—that would be easier to handle. It fails subtly, intermittently, and in ways difficult to predict or prevent.

Key Takeaways

1. **Hallucinations are now embedded in workflows.** Build verification at every critical checkpoint.
2. **Inconsistency requires explicit workflow definitions.** Don't assume the same agent will always take the same path.
3. **Over-autonomy is dangerous.** Define clear scopes and permission systems.
4. **More tools = more confusion.** Limit tool sets and validate calls carefully.
5. **Long context used context.** Place critical information at the start or end.
6. **Multi-agent adds coordination complexity.** Only use when truly needed.
7. **Invest in observability.** Debugging without proper tooling is guesswork.

Part 6: The Road Ahead

What practitioners should expect and prepare for.

The Boring Infrastructure Wins

The most effective AI work in 2026 will be boring infrastructure and data work.

It's going to get the most effectiveness out of these agents: - Looking at your data from first principles - Understanding your application architecture - Building foundational work instead of chasing shiny new models

AI will become a critical part of infrastructure—not just in terms of spend, but in how you can do well as a company and how you use that power to improve your services and building speed.

Talent Profiles Are Evolving

We're seeing reduced boundaries between roles: - More focus on getting the best out of your previous experience - Learning about different parts of product deployment - Bringing all of those pieces together

The future isn't AI specialists working in isolation—it's AI capabilities embedded across the organization.

AI as Standard Infrastructure

It's not just about being an AI-native company or building an AI-enabled product.

There are many ways every person can benefit from AI: - Product research - Writing - Building apps for personal use cases

There's competitive advantage in: - How you actually use AI - Whether you understand the rough edges - That tribal knowledge and context becoming your moat

Integration Quality Beats Model Selection

Models are doing very similar performance on benchmarks.

What matters: - How well you integrate models into your workflows - How you handle your specific use cases - How you build evaluation and monitoring - How you develop organizational AI capabilities

The ease of integration and real-world performance is determined not by benchmarks, but by how it's defined and developed for your own use case.

Picking the “best” model matters less than building the best systems around whatever model you choose.

Building Evaluation is Non-Negotiable

Understanding what your agent is doing—not just building it, but understanding what use cases you're building it for—is extremely important.

This is going to be a key factor in terms of AI ROI: - What business outcomes can be improved? - How do you measure ROI on that?

Teams that can measure and demonstrate value will keep investing. Teams that can't will see budgets cut.

The Long Game

AI's value doesn't come from a single breakthrough or incredible benchmark performance that you plug in and everything starts working.

It's this slow grind—but an extremely fruitful process of: - Understanding current capabilities of models - Understanding the hardinesses and tools you're building - Building not just for current capabilities but always placing the future in mind - Priming your company so new breakthroughs can be seamlessly incorporated

What Success Looks Like

The organizations that succeed will be those that: - Treat AI not as magic, but as engineering - Build reliable systems over clever ones - Invest in boring infrastructure - Create value for users at sustainable costs

The most impressive deployments won't be the most technically sophisticated—they'll be the ones solving real problems for real users.

Conclusion

2025 was the year AI applications grew up. Models became more capable, tools became more mature, and practitioners learned—often the hard way—what actually works in production.

Key Lessons

1. **Context engineering > prompt tricks.** Design the entire information environment, not just the initial prompt.
2. **Reasoning models trade speed for reliability.** Use them for complex tasks where correctness matters.
3. **RAG evolved, it didn't die.** Structure, agency, and multimodal parsing made retrieval more powerful.
4. **Agents achieved T-shaped success.** Broad automation of simple tasks, deep impact in specific verticals.
5. **New capabilities brought new failure modes.** Subtle hallucinations, inconsistent reasoning, over-autonomy.
6. **Evaluation is the bottleneck.** You can't improve what you can't measure.
7. **Infrastructure beats heroics.** Boring, reliable systems outperform clever, fragile ones.

The Bottom Line

AI's value doesn't come from single breakthroughs that magically solve everything. It comes from the slow grind of integration, evaluation, and incremental improvement.

The organizations that succeed will be those that treat AI not as magic, but as engineering—with all the discipline that implies.

Acknowledgements

This report was authored by **Aishwarya Naresh Reganti** (LevelUp Labs) and **Kiriti Badam** (OpenAI), based on a live presentation to 2000+ practitioners.

Copyright 2026 Aishwarya Naresh Reganti & Kiriti Badam. All rights reserved.

End of Report