From Prompt Engineering to Intent Engineering: Why AI Agents Need More Than Context

February 24, 2026 · by Mark Bartlett

Download this guide as PDF

Quick Answer: Prompt engineering tells AI what to do. Context engineering (RAG, MCP, memory systems) tells AI what to know. Neither tells AI what to want. Intent engineering encodes goals, values, tradeoffs, and decision boundaries so agents optimize for what you actually need, not just what they can measure. Klarna's AI handled 2.3 million conversations a month and cut resolution time from 11 minutes to 2. It also destroyed customer trust and forced the CEO to publicly admit cost-cutting via AI had gone too far. The agent had prompts and context but no intent — no judgment about when to bend policy, when to escalate, when efficiency should yield to generosity. This is the gap that explains why 60% of companies see little or no value from AI (BCG), why 40% of agentic AI projects will be canceled by 2027 (Gartner), and why only 5% of Microsoft Copilot pilots scaled to full deployment. Local AI builders have a structural advantage here: persistent state, private value frameworks, and full control over alignment tuning.

📚 **Related:** Building AI Agents with Local LLMs · Context Rot and the Forgetting Fix · Session-as-RAG Memory · Function Calling with Local LLMs · Planning Tool

Klarna’s AI assistant handled 2.3 million customer service conversations per month. It cut resolution time from 11 minutes to under 2. It did the work of 700 full-time agents and saved the company $60 million.

In May 2025, CEO Sebastian Siemiatkowski went on Bloomberg and said the AI strategy had gone too far. “As cost unfortunately seems to have been a too predominant evaluation factor,” he said, “what you end up having is lower quality.” Klarna started rehiring humans.

Both things were true at the same time. The AI succeeded at the metric it was given (speed, volume, cost) and failed at what the company actually needed (trust, judgment, knowing when to spend extra time with an angry customer). The system had prompts. It had context. What it didn’t have was intent.

This matters for anyone building local AI agents, because the same failure mode is waiting for you at smaller scale.

The three disciplines

Credit to Nate Jones (AI News & Strategy Daily) for crystallizing this progression in a February 2026 essay that used the Klarna story as its centerpiece.

Prompt engineering: “How do I talk to AI?”

This is where most people start. You craft a system prompt, adjust temperature, experiment with few-shot examples, and try to coax the right output from a model. It works for individual sessions. It’s how you get ChatGPT to write a cover letter or get Ollama to format JSON.

Prompt engineering is session-scoped. Each conversation starts from zero. There’s no memory, no accumulated knowledge, no persistence. For one-shot tasks, that’s fine. For agents that need to operate over hours, days, or weeks, it’s not enough.

Shopify CEO Toby Lutke posted in June 2025: “I really like the term ‘context engineering’ over prompt engineering. It describes the core skill better: the art of providing all the context for the task to be plausibly solvable by the LLM.” Andrej Karpathy endorsed the same shift around the same time.

Context engineering: “What does AI need to know?”

Anthropic’s engineering team published their context engineering framework in September 2025, developed from building Claude Code. The core idea: stop obsessing over prompt wording and start thinking about what information enters the model’s attention window at each inference step.

Context engineering includes:

- RAG retrieval to pull relevant documents
- MCP connections to give models access to tools and data sources
- Session memory that persists across conversations
- Context compaction that summarizes history as conversations grow
- Multi-agent architectures where specialized sub-agents return condensed results

It works. Anthropic reported that combining their memory tools with context editing improved agent performance by 39% over baseline and reduced token consumption by 84% in 100-turn dialogues.

Google's Agent Development Kit (released April 2025) formalized the architecture into four layers: session state (current conversation), persistent memory (across sessions), working context (what the agent sees right now), and artifacts (files and data produced along the way).

Context engineering is necessary. It's a massive improvement over raw prompting. But it has a ceiling.

Intent engineering: "What does AI need to want?"

Jones defines intent engineering as "the discipline of making organizational purpose — goals, values, tradeoffs, decision boundaries — machine-readable and machine-actionable so that when you deploy an autonomous system, it optimizes for what your company actually needs, not just what it can measure."

This is the layer Klarna was missing. Their AI had context (customer records, order history, policy documents) and good prompts (resolve the ticket efficiently). What it lacked was judgment: when to bend policy for a loyal customer, when to spend 15 minutes instead of 2 because the person is upset, when efficiency should lose to generosity.

Those aren't context problems. You can stuff a million tokens of policy documents into a context window and the model still won't know that right now, with this customer, the correct move is to eat the cost and issue a full refund without argument. That requires values, not information.

The Klarna story in full

The timeline matters because it shows how fast things went wrong.

January 2024: Klarna launches its AI assistant globally, powered by OpenAI. Within the first month it handles two-thirds of all customer service chats.

February 2024: Public announcement. 2.3 million conversations. Resolution time down from 11 minutes to under 2. 25% drop in repeat inquiries. "Customer satisfaction on par with human agents." Estimated $40M profit improvement.

2024-2025: Headcount falls from ~5,500 to ~3,400 through natural attrition and a hiring freeze. The AI is doing the work of 853 employees. The cost savings hit $60M.

Early 2025: Internal reviews find the AI lacks empathy and can't handle refund negotiations or nuanced multilingual support. Customers describe responses as generic and repetitive.

May 2025: Siemiatkowski tells Bloomberg the company went too far. Klarna begins rehiring humans for a hybrid model where AI handles basic inquiries and humans handle anything requiring discretion.

The most telling detail: Klarna's Q1 2025 earnings report claimed "no drop in consumer satisfaction." Their CSAT metric showed 4.4/5 for AI versus 4.2/5 for humans. The AI was scoring higher than humans on the satisfaction survey while simultaneously destroying enough customer trust to force a public reversal.

They were measuring the wrong thing. The survey captured "was your issue resolved?" but not "do you trust this company?" or "will you come back?" The intent gap isn't just about the model lacking judgment. It's about the organization not encoding the right goals.

Why context engineering hits a ceiling

If you've been building local AI agents, you've probably run into some version of this already.

Stuffing context windows doesn't give agents judgment

Every token you add to the context window competes for the model's attention. A 32B model with 128K context can hold a lot of information. But more information doesn't produce better decisions — it often produces worse ones, because the model treats policy documents and customer sentiment with equal weight. This is context rot at an architectural level.

MCP connects tools but doesn't encode values

MCP is great for giving agents access to databases, APIs, and file systems. It solves the "how does the agent do things?" problem. It doesn't touch the "should the agent do this particular thing right now?" problem.

An agent with MCP access to your email, calendar, and Slack can take actions. Whether it should reschedule your meeting, send that message, or just wait depends on priorities that aren't in the tool schema.

RAG retrieves knowledge but not wisdom

RAG gives your agent access to documents, past conversations, and reference material. It answers "what has been said about this?" It doesn't answer "what matters most here?" or "how should I weigh these conflicting priorities?"

The enterprise numbers confirm this

The pattern holds across the industry:

Source	Finding
BCG (Sep 2025)	60% of companies report little or no value from AI despite major investment
McKinsey (Mar 2025)	80%+ see no EBIT impact from generative AI specifically
Gartner (Jun 2025)	Over 40% of agentic AI projects will be canceled by 2027
Gartner (2025)	Only 5% of Microsoft Copilot pilots scaled to full deployment
Deloitte (2026)	74% of organizations hope to grow revenue through AI; only 20% actually are

These companies have budgets, talent, and access to the best models. What they don't have is a way to encode organizational intent into autonomous systems.

What intent engineering actually requires

This is the hard part, and the field is early. But the outlines are becoming clear.

Machine-readable goal structures

"Increase customer satisfaction" is not a goal an agent can act on. It's a sentiment. Intent engineering requires decomposing that into specific, measurable, machine-actionable structures:

- What signals indicate the customer needs extra attention? (repeated contacts, escalation language, account tenure)
- What actions are available at each decision point? (standard resolution, expedited resolution, escalation to human, policy exception)
- What tradeoffs are acceptable? (spend up to $50 to retain a customer with >2 years of history; escalate immediately for any legal language)
- Where are the hard boundaries? (never promise something the system can't deliver; always disclose that this is an AI)

Pawel Huryn's intent engineering framework (published January 2026) breaks this into seven components: objective, desired outcomes, health metrics, strategic context, constraints, decision types with autonomy levels, and stop rules. The stop rules matter — they define when the agent should hand off to a human or do nothing at all.

Delegation frameworks

Not every decision should be autonomous. Intent engineering means defining clear boundaries: what the agent can decide alone, what needs human approval, and what it should never attempt.

This is where most agent deployments fail. They give the agent access to tools and assume the model's judgment is sufficient. Then the model sends an email that shouldn't have been sent, or schedules a meeting during a conflict, or issues a refund that violates policy in a way the customer will exploit.

Feedback loops that measure alignment

If your agent is optimizing for the wrong metric (like Klarna optimizing for resolution speed), you need to catch that before it becomes a public reversal. This means measuring not just "did the agent complete the task?" but "did the agent's behavior align with our values?"

Drift detection matters here. An agent that starts well can drift over time as its memory fills with patterns that reinforce the wrong objective. Context rot isn't just a technical problem — it's an alignment problem.

Why local matters for intent engineering

This is where the article becomes relevant to what you're probably building.

Persistent state

Intent requires memory that accumulates over time. Not just what happened in this session, but what the agent learned from hundreds of past interactions. Cloud API calls are stateless — each request starts fresh. Local agents running on your hardware can maintain persistent memory, episodic logs of past decisions, and evolving models of what works and what doesn't.

Google's ADK separates session state from long-term memory for exactly this reason. But with cloud APIs, that long-term memory still lives on someone else's infrastructure. With local agents, you own it entirely.

Privacy

Your organizational values, decision frameworks, and delegation boundaries are sensitive IP. The tradeoffs encoded in your intent layer reveal your business priorities, your risk tolerance, and your competitive strategy. Shipping that to a cloud API means trusting a third party with the logic that drives your most important decisions.

Local keeps it on your hardware. Nobody sees your intent layer except you.

Control

When an agent's behavior drifts from your intent, you need to diagnose and fix it. With local systems, you can inspect every layer: the model weights, the context window contents, the memory store, the decision logs. You can tune, retrain, or restructure any component.

With cloud APIs, you can adjust your prompt and hope the next model update doesn't change the behavior you tuned for. (OpenAI retired GPT-4o on February 13, 2026 with minimal notice. Every agent built on GPT-4o's specific behaviors had to be re-evaluated.)

Latency

Judgment calls often happen mid-conversation. An agent deciding whether to escalate a customer issue or bend policy shouldn't wait 200ms for a round trip to a datacenter. Local inference gives sub-millisecond network latency for the decision layer, which matters when the agent needs to make dozens of micro-decisions per interaction.

Cognitive architecture: beyond briefing packets

Here's the difference between context engineering and intent engineering in concrete terms.

Context engineering gives an agent a briefing packet before each task. "Here's the customer's history, here's our policy, here's the current queue depth." The agent reads the briefing and acts.

Intent engineering gives an agent something closer to a lifetime of experience. Not just what to know, but how to weigh competing priorities, when to break from standard procedure, and what "doing good work" means in this specific organizational context.

For local AI agents, this translates to concrete architectural choices:

An identity layer gives the agent a persistent configuration of its role, values, and decision style. Not a system prompt that reloads each session, but a stable identity that shapes behavior over time.

Episodic memory stores experiences, not just facts. Not “customer X bought product Y” but “last time I bent the refund policy for a long-term customer, it led to a positive review and a repeat purchase.” This is how humans develop judgment — by accumulating outcomes, not just data.

Decision journaling means the agent logs not just what it did, but why it chose that action over alternatives. Over weeks, these logs become a corpus that future decisions can draw on. “What did I do last time I faced this tradeoff?”

And timing gates. Sometimes the right action is inaction. An agent that reflexively responds to every input is missing the judgment to wait, observe, and act at the right moment. This is different from a cooldown timer — it’s a model of when intervention creates value versus when it creates noise.

Practical takeaways for local AI builders

If you’re running local agents with Ollama or building custom agent loops, here’s where to start.

Don’t just wire up tools — encode what your agent should optimize for

MCP and function calling give your agent hands. Intent gives it priorities. Before connecting your agent to email, databases, or APIs, write down: what should this agent optimize for? What should it never do? What tradeoffs are acceptable?

Put those in a structured format the agent can reference — not buried in a paragraph of system prompt, but as explicit decision rules.

Build memory systems that learn from past decisions

Session-as-RAG stores conversation history. Go further: store decision outcomes. When the agent makes a choice, log the context, the decision, and (when possible) the result. Over weeks and months, this becomes a corpus of judgment the agent can draw on.

Define decision boundaries before giving agents autonomy

Write explicit rules: the agent can do X without asking, needs approval for Y, and should never attempt Z. Review these boundaries weekly as you learn what the agent handles well and where it misjudges.

Start small

One workflow. Clear intent. Tight feedback loop. Let the agent handle customer support triage with three explicit priority rules and a hard escalation boundary. Measure not just “did it complete the task” but “did it make the right call?” Expand autonomy only as alignment evidence accumulates.

Use the Planning Tool to size your hardware

Intent-aware agents need larger models. Decision quality correlates with model size — a 7B model can follow instructions, but it struggles with the nuanced tradeoff reasoning that intent engineering requires. Budget for 32B minimum, 70B if your decisions have real consequences. Plan your VRAM accordingly.

Where this goes

Prompt engineering dominated 2023-2024. Context engineering dominated 2025. Intent engineering is where 2026 is heading.

The companies that figure out how to encode organizational purpose into autonomous systems will get the ROI that 60% of enterprises are currently missing. The ones that keep stuffing context windows and hoping for judgment will keep building faster versions of Klarna’s original mistake — AI that succeeds at the wrong thing.

For local AI builders, the advantage is structural. You own the state and the memory. You can iterate on the intent layer without waiting for a vendor to update their platform or worrying that your competitive logic is training someone else’s model.

The tools exist. Agents, function calling, memory systems, context management. What’s missing — for most people — is the deliberate work of encoding what the agent should want.

That’s the next layer. And it’s yours to build.

Source: https://insiderllm.com/guides/intent-engineering-ai-agents/

Free guides for running AI locally