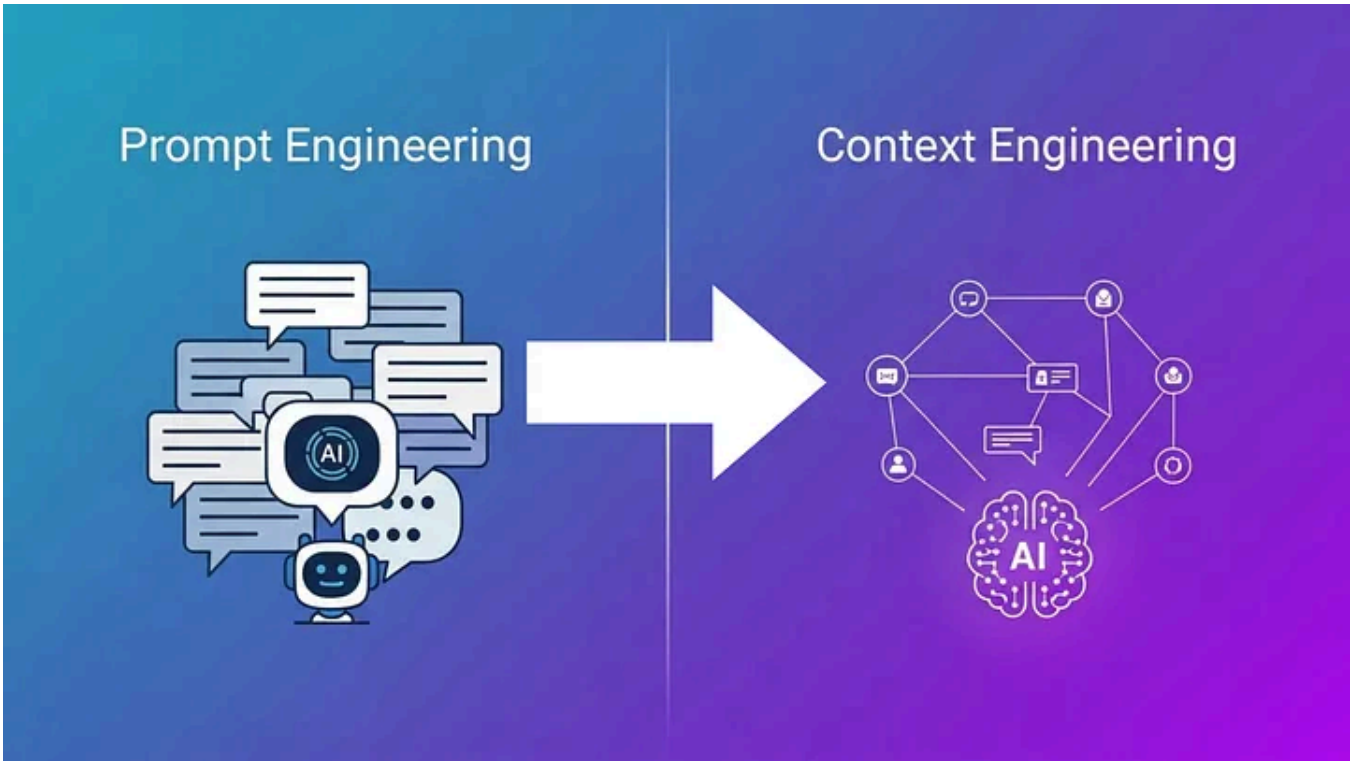
Member-only story

How These 3 Articles Completely Changed the Way I Build AI Agents

9 min read · Oct 15, 2025

Avijnan Chatterjee Follow

The only three reads you need to master context engineering and take your AI agents to the next level



If you are not a member, you can read the story here.

Prompt engineering is so yesterday.

If you're still obsessing over the perfect system prompt while building AI agents, you're missing the bigger picture.

To be very honest with you — when I first started exploring AI agents, I thought I had the entire workflow figured out. After all, I'd spent months writing perfect prompts, learning about few-shot examples, and understanding how to structure instructions.

But then my agents started failing in ways that better prompts was not able to fix.

They'd lose track of their goals after 50 tool calls. They'd repeat the same mistakes. They'd blow through context limits faster than I could say "*token budget.*"

That's when I discovered that building agents isn't about prompt engineering anymore.

It's about something called **Context Engineering**.

What is Context Engineering, Anyway?

Try to visualize this: If prompt engineering is about writing the perfect instruction manual, context engineering is about managing everything your agent sees, remembers, and forgets throughout its entire journey.

As agents take on more complex tasks — debugging codebases, conducting research, managing long conversations — they generate massive amounts of information. The question isn't just "what should I tell my agent?" but "what should my agent know at *this exact moment* to take the next best action?"

This shift is fundamental. And honestly, when I started digging into this topic, I got overwhelmed.

There are dozens of blog posts, research papers, and techniques scattered across the internet. Some contradicted each other. Others were too theoretical to be useful.

So I did what I have doing quite often now — ran deep research in Google Gemini and I came across three revolutionary articles.

Three articles from the biggest names in AI agents teach you everything you need to know about context engineering.

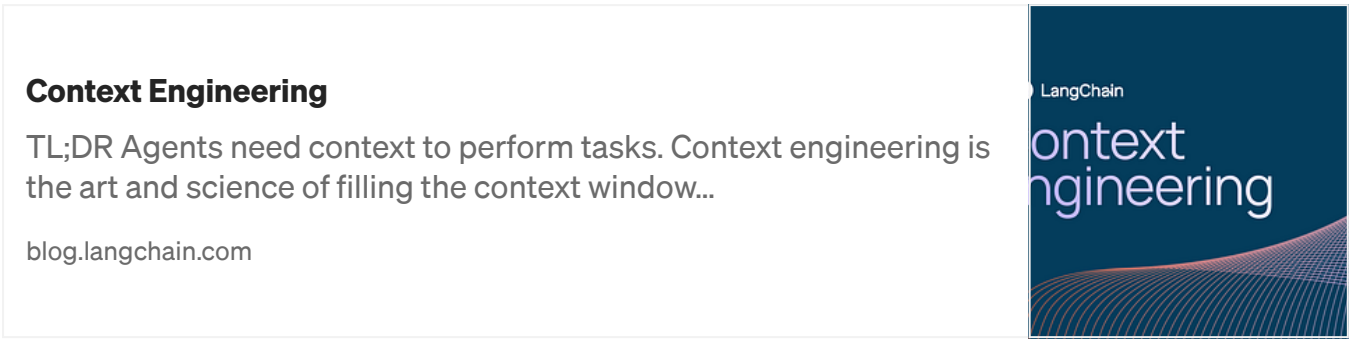
I'm talking about **LangChain** (the framework powering countless AI applications), **Anthropic** (the folks behind Claude), and **Manus** (a production agent handling millions of users).

Let me walk you through why these three articles, in this specific order, will transform how you think about building agents.

Article 1: LangChain — The Mental Model

Start with LangChain because it gives you the clearest mental model and framework for making sense of context engineering.

You can access the article from Langchain by clicking below —



The Perfect Analogy

LangChain opens with an analogy that makes everything click:

> *LLMs are like a new kind of operating system. The LLM is like the CPU and its context window is like the RAM, serving as the model's working memory.*

This single comparison changed how I think about agents.

Just like RAM has limited capacity, your context window needs active management. And just like an operating system curate what fits into RAM, context engineering manages what enters your agent's working memory.

The Four-Strategy Framework

LangChain organizes context engineering into four clear strategies that become your mental buckets:

1. Write Context — Saving information outside the context window

- **Scratchpads:** Temporary notes during task execution

- **Memories:** Long-term storage across sessions (think user preferences, learned behaviors)

2. Select Context — Pulling information into the context window

- Use RAG for knowledge retrieval
- Apply semantic search for tool selection
- Retrieve relevant memories when needed

3. Compress Context — Retaining only necessary tokens

- **Summarization:** Distill long trajectories into key points
- **Trimming:** Remove older, less relevant messages

4. Isolate Context — Splitting context across components

- **Multi-agent:** Each sub-agent has its own focused context window
- **Sandboxing:** Keep token-heavy objects isolated in execution environments

State objects: Store context separately from what the LLM sees

Why Context Fails

LangChain references Drew Breunig's breakdown of context failure modes that you need to watch for:

- **Context Poisoning:** When a hallucination contaminates the context
- **Context Distraction:** When context overwhelms the training
- **Context Confusion:** When superfluous context influences responses
- **Context Clash:** When parts of the context disagree

Understanding these failure modes helps you design defenses against them.

Why read this article first?

Because LangChain gives you the organizing framework.

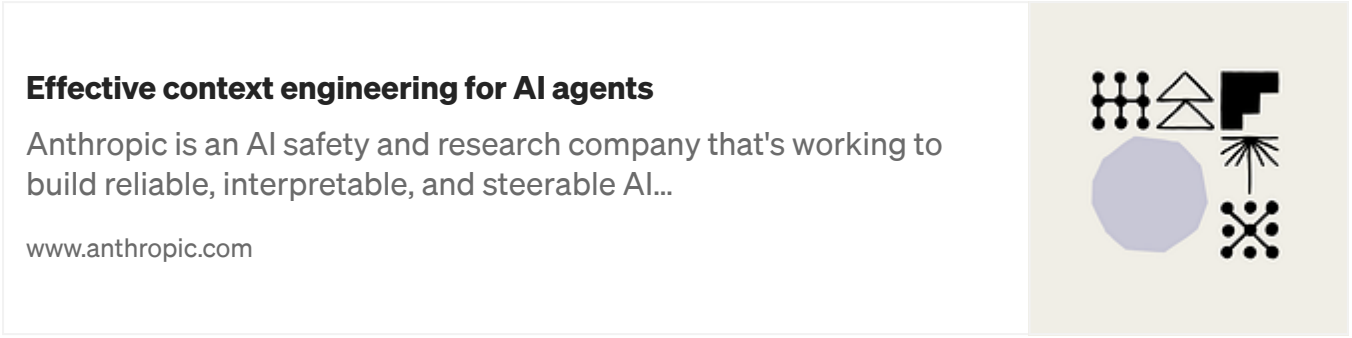
Once you have these four buckets in your head — ***write, select, compress, isolate*** — everything else you learn has a place to go.

Without this structure, the details from other articles can feel overwhelming.

Article 2: Anthropic — The Foundation

Now that you have the framework from LangChain, Anthropic teaches you *why* context engineering matters at a fundamental level and gives you the principles to guide your decisions.

You can access the article from Anthropic by clicking below —



The Core Problem: Context Rot

Anthropic introduces the concept of **"context rot"** — as you add more tokens to your context window, your model's ability to recall information actually decreases. It's not a hard limit; it's attention fatigue.

Here's why: LLMs use the transformer architecture, which means every token attends to every other token. That creates n^2 pairwise relationships. The more tokens you add, the thinner that attention gets stretched.

Context, therefore, must be treated as a finite resource with diminishing marginal returns.

The Guiding Principle

Anthropic gives you the one principle that should drive all your context engineering decisions:

Find the *smallest possible set of high-signal tokens* that maximize the likelihood of your desired outcome.

This means every token in your context window should justify its existence. It's not about having more information — it's about having the *right* information.

The "Right Altitude" for Prompts

One concept that really clicked for me was their idea of the **"right altitude"** for system prompts. There are two failure modes:

1. **Too prescriptive**: Hardcoding brittle if-else logic that breaks easily
2. **Too vague**: Assuming shared context that doesn't exist

The sweet spot? Specific enough to guide behavior, flexible enough to let the model reason intelligently.

Just-in-Time Context: The Game Changer

Here's where Anthropic really challenged my thinking: they advocate for **"just-in-time" context** over pre-loading everything upfront.

Instead of retrieving all potentially relevant data before your agent starts (the traditional RAG approach), maintain lightweight identifiers — file paths, URLs, queries — and let your agent dynamically load what it needs *when it needs it*.

This mirrors how humans work. We don't memorize entire databases; we remember where to find information.

Claude Code uses this hybrid approach beautifully: it pre-loads CLAUDE.md files for quick access but then uses commands like grep and glob to explore the codebase just-in-time.

Long-Horizon Strategies

For tasks spanning hours and thousands of turns, Anthropic outlines three techniques:

1. **Compaction**: Summarize context when hitting limits (Claude Code auto-compacts at 95% capacity)
2. **Structured note-taking**: Agents write persistent notes outside the context window
3. **Sub-agent architectures**: Specialized agents handle focused tasks, returning only condensed summaries

The example of Claude playing Pokémon fascinated me — the agent maintains precise tallies across thousands of game steps, builds maps, and remembers strategies. After context resets, it just reads its own notes and continues.

Why read this article second?

Because after having the framework from LangChain, you need the *why* and the guiding principles.

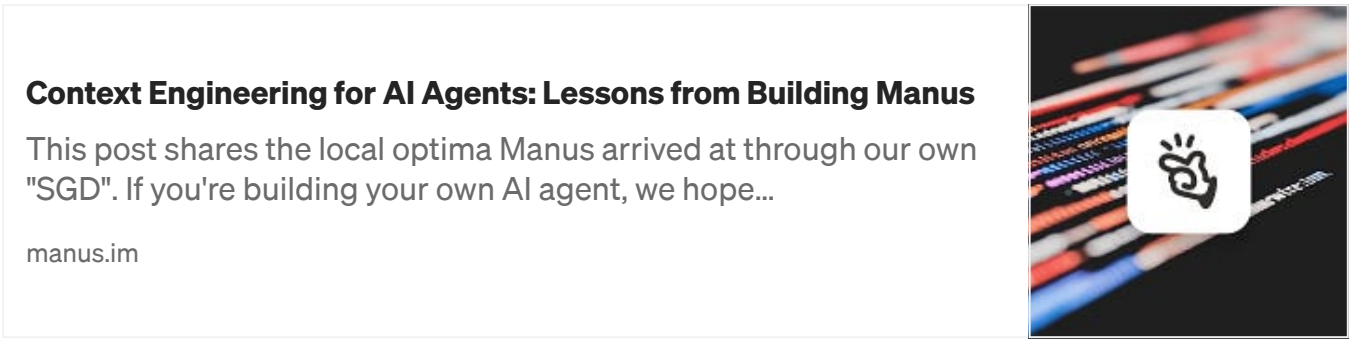
Anthropic shows you the fundamental constraints (attention budget, context rot) and gives you decision-making principles that work across all four LangChain strategies.

Article 3: Manus — The Reality Check

After understanding the framework and principles, Manus shows you what actually matters in production. This team rebuilt their agent framework *four times* while serving millions of users.

They call their process "***Stochastic Gradient Descent***" — a tongue-in-cheek reference to the manual, experimental nature of figuring out what works.

You can access the article from Manus by clicking below —



The KV-Cache Obsession

Here's what immediately grabbed my attention: Manus argues that **KV-cache hit rate is the single most important metric** for production agents.

Why? Because cached input tokens cost $0.30/MTok with Claude Sonnet, while uncached ones cost $3/MTok. That's a **10x difference**.

This completely reframed how I design context. It's not just about what information to include — it's about *how you structure it to maximize cache reuse*.

Three critical practices:

1. **Keep your prompt prefix stable**: Even a timestamp at the beginning invalidates your entire cache
2. **Make context append-only**: Never modify previous actions or observations

3. **Mark cache breakpoints explicitly**: Account for cache expiration

Mask, Don't Remove

Here's a counterintuitive insight: When dealing with too many tools, don't dynamically add and remove them. **Mask them instead.**

Why? Because dynamically changing tool definitions invalidates your cache and confuses the model when it sees references to tools that no longer exist.

Manus uses a state machine to control which tools are available, masking token logits during decoding rather than modifying context.

The File System as Ultimate Context

This idea really stuck in my head: **Treat your file system as context.**

Manus argues that aggressive compression inevitably loses information. But you can't predict which observation from ten steps ago might become critical later.

So instead, they use the file system as unlimited, persistent and directly operable memory. The agent writes to and reads from files on demand. Their compression strategies are always *restorable* — a webpage's content can be dropped as long as the URL remains.

Keep the Wrong Stuff In

This completely goes against conventional wisdom: **Don't hide errors. Leave them in context.**

When your agent sees a failed action and the error message, it implicitly updates its beliefs and avoids repeating the same mistake. Erasing failures removes evidence. Without evidence, the model can't adapt.

Don't Get Few-Shotted

Here's another counterintuitive lesson: **Too much consistency in your context can backfire.**

LLMs are excellent mimics. If your context is full of similar action-observation pairs, the model will follow that pattern even when it's no longer optimal. Manus saw this when reviewing 20 resumes — the agent fell into a rhythm, repeating similar actions.

The fix? **Introduce controlled diversity.** Vary serialization templates, use alternate phrasing, add minor noise to formatting.

Why read this last?

Because Manus shows you what the framework and principles look like when rubber meets road.

You learn about the 10x cost difference from caching, the file-system-as-context approach, and why keeping errors visible actually improves performance. These are lessons you only learn by shipping to millions of users.

How These Three Articles Work Together

Here's why this sequence matters (according to me) —

1. **LangChain** gives you the *framework* — four clear buckets (write/select/compress/isolate) and the CPU/RAM mental model. You now have structure.
2. **Anthropic** gives you the *principles* — why context is finite (context rot, attention budget) and how to make decisions (smallest high-signal token set, right altitude, just-in-time). You now have decision-making criteria.
3. **Manus** gives you *production reality* — KV-cache optimization, file-system memory, keeping errors visible, preventing pattern lock-in. You now have battle-tested tactics.

Together, they cover the full spectrum from mental models to guiding principles to production engineering.

Final Thoughts

If you're building AI agents — whether you're just starting out or already have something in production — these three articles are required reading.

LangChain will give you clarity.

Anthropic will give you principles.

Manus will save you from expensive mistakes.

The shift from prompt engineering to context engineering is real, and it's happening now. Agents that master context management will be faster, cheaper, and more capable. Agents that don't will hit walls you can't prompt your way out of.

I've read dozens of articles on this topic. These three are the ones I keep coming back to.

Do yourself a favor: Block out three hours, grab some coffee, and read them in order. Your future self (and your token budget) will thank you.

. . .

Have you started thinking about context engineering for your agents? What challenges are you facing? Drop a comment below — I'd love to hear about your experience.

Note: This article synthesizes insights from three publicly available blog posts. All credit goes to the respective authors and companies. I'm just sharing what I've learned along the way.

A message from our Founder

Hey, Sunil here. I wanted to take a moment to thank you for reading until the end and for being a part of this community.

Did you know that our team run these publications as a volunteer effort to over 3.5m monthly readers? **We don't receive any funding, we do this to support the community.** ❤️

If you want to show some love, please take a moment to **follow me on LinkedIn, TikTok, Instagram**. You can also subscribe to our **weekly newsletter**.

And before you go, don't forget to **clap** and **follow** the writer!

Context Engineering | Ai Agent | Langchain | Prompt Engineering | Agentic Ai

Follow

Published in Artificial Intelligence in Plain English

31K followers · Last published 12 hours ago

New AI, ML and Data Science articles every day. Follow to join our 3.5M+ monthly readers.

Follow

Written by Avijnan Chatterjee

13 followers · 12 following

I write about AI and emerging tech — in plain English. Helping curious minds make sense of the tools shaping our future. 🧠 Youtube: @aifuturisttoolbox

No responses yet

Bgerby

What are your thoughts?

More from Avijnan Chatterjee and Artificial Intelligence in Plain English

In Generative AI by Avijnan Chatterjee

This Free AI Tool Does What ChatGPT Can’t

Discover Google AI Studio’s hidden features: system prompts, Compare Mode, and real-time screen sharing. Free tutorial for analyzing...

3d ago 41

In Artificial Intelligence in Plain English by Simranjeet Singh

RAG is Hard Until I Know these 12 Techniques → RAG Pipeline to 99% Accuracy

RAG is Hard Until I Know these 12 Techniques → RAG Pipeline to 99% Accuracy. Best Blog to Scale or increase RAG Pipelines Accuracy.

Sep 27 476 8

In Artificial Intelligence in Plain English by Simranjeet Singh

OpenAI ML Engineer Interview Questions 2025

A mock interview with an OpenAI ML engineer covering LLM deployment, low-latency inference, quantization, mixed precision, and strategies.

Sep 24 140 7

In Generative AI by Avijnan Chatterjee

How I Create Apps and Images in Minutes (Free)

Generate professional images with accurate text using Imagen 4, realistic Voiceovers, and build functional apps without coding. Complete...

2d ago 51

See all from Avijnan Chatterjee

See all from Artificial Intelligence in Plain English

Recommended from Medium

Pawel

Claude Skills: The AI Feature That Actually Solves a Real Problem

Yesterday, Anthropic quietly released what might be the most practical AI feature of 2025. It's not flashier models or better benchmarks...

6d ago 142 8

In Level Up Coding by Fareed Khan

Building an Agentic Deep-Thinking RAG Pipeline to Solve Complex Queries

Planning, Retrieval, Reflection, Critique, Synthesis and more

3d ago 635 3

Bytefer

A 0.9B Open-Source Model for SOTA Document Parsing: Outperforming GPT-4o and Gemini 2.5 Pro

Beyond OCR: Parsing text, tables, formulas & charts in 109 languages. Get SOTA document parsing that runs locally.

6d ago 53 3

Mahimai Raja J

FastRTC: Zero To One | Developer Edition

A Complete Guide to Understand and Implement FastRTC Efficiently.

Jul 16 21 1

In CodeToDeploy by TechToFit - Master Your Life with Tech

I Tried Google's New AI Agents. It's a Gold Rush.

I spend my days deep in the world of AI, but every so often, something drops that makes me stop everything. This is one of those times...

Oct 10 123 3

In Coding Nexus by Civil Learning

MarkItDown: Convert Anything into Markdown—the Smart Way to Feed LLMs

You know that feeling when you're trying to feed a PDF or a Word document into an LLM, and it just doesn't understand what's going on...

Oct 15 140 3