Breaking Down Context Engineering

Breaking Down Context Engineering

All you need to know about the challenges surrounding the practice.

The topic that has been dominating headlines in the AI agent world for the past months — context engineering. This article outlines practical notes from building agentic systems. We will look at all types of context that agentic systems rely on and the challenges that come with managing it.

For engineers who have been building AI agents, context engineering is a new name but not a new practice — it is how we guided agents to perform the work in the first place. The "old school" prompt engineering is a subset of context engineering.

What is Context Engineering and Why It Matters

In simple terms, an agentic system is a topology of LLM calls connected via different patterns. Each output of an LLM node influences the downstream system.

The quality of an agent is only as good as the context you pass into prompts at each step. Bigger context windows are not a silver bullet. Pushing too much data creates issues like:

- **Context poisoning** — a hallucination makes it into context
- **Context distraction** — context overwhelms training
- **Context confusion** — superfluous context influences the response
- **Context clash** — parts of the context disagree

Context engineering aims to provide the minimal, focused context at each step so the agent can do its job.

Below are common context types and challenges.

Instructions — System Prompt

Where it fits: Rails that define role, behavior, boundaries before user input.

Challenges:

- Alignment within limited space
- Prompt injection risk
- Conflicts with user needs vs policy

User Prompt

Where it fits: The user's direct request.

Challenges:

- Multi-turn, multi-step tasks
- Break down into sub-tasks and chain prompts
- Maintain intent over turns with evals

Retrieved Context

Where it fits: External info via RAG or APIs, injected into prompts.

Challenges:

- Find and rank the right snippets
- Context window limits and confusion modes
- Corpus prep — chunking, metadata, quality

State (Short-Term Memory)

Where it fits: Working memory and selective conversation history.

Challenges:

- Finite window — summarise, clip, cache
- Context drift — re-inject key instructions

Long-Term Memory

Where it fits: Persisted knowledge — preferences, facts, past interactions.

Challenges:

- Relevance and retrieval
- Consistency, updating, privacy
- Integrate without confusion

The Connection Between Long and Short-Term Memory

1. **Episodic** — past interactions and actions in vectors
2. **Semantic** — external and internal knowledge for grounding
3. **Procedural** — system prompt, tools, guardrails

4. Pull from long-term when needed
5. Compiled working memory becomes the prompt

We label 1–3 long-term memory and 5 short-term memory.

Tools

Where it fits: External capabilities — APIs, code, search, retrieval.

Vanilla Tool Use (Native Function Calling)

1. User query into agent
2. Functions/tools defined (procedural)
3. List of tools + query to LLM
4. Agent executes functions
5. Results sent back to LLM
6. Final answer returned

Challenges:

- Knowing when and how to call tools
- Integrating tool outputs without blowing context
- Maintaining state across reason-act loops
- Error handling and security guardrails

Prompt format example:

```
when making tool calls, use this exact format:
{
  "name": "tool_name",
  "arguments": {
    "parameter1": "value1",
    "parameter2": "value2"
  }
}
```

Structured Output

Where it fits: Enforce formats like JSON, HTML, SQL for downstream steps.

Challenges:

- Reliability and exact formatting
- Use schemas, grammars, validators and retrying

Wrapping Up

Context engineering is evolving fast. Practices will change as tooling and models change. Start small, measure, and iterate. Happy building.
