Context Engineering

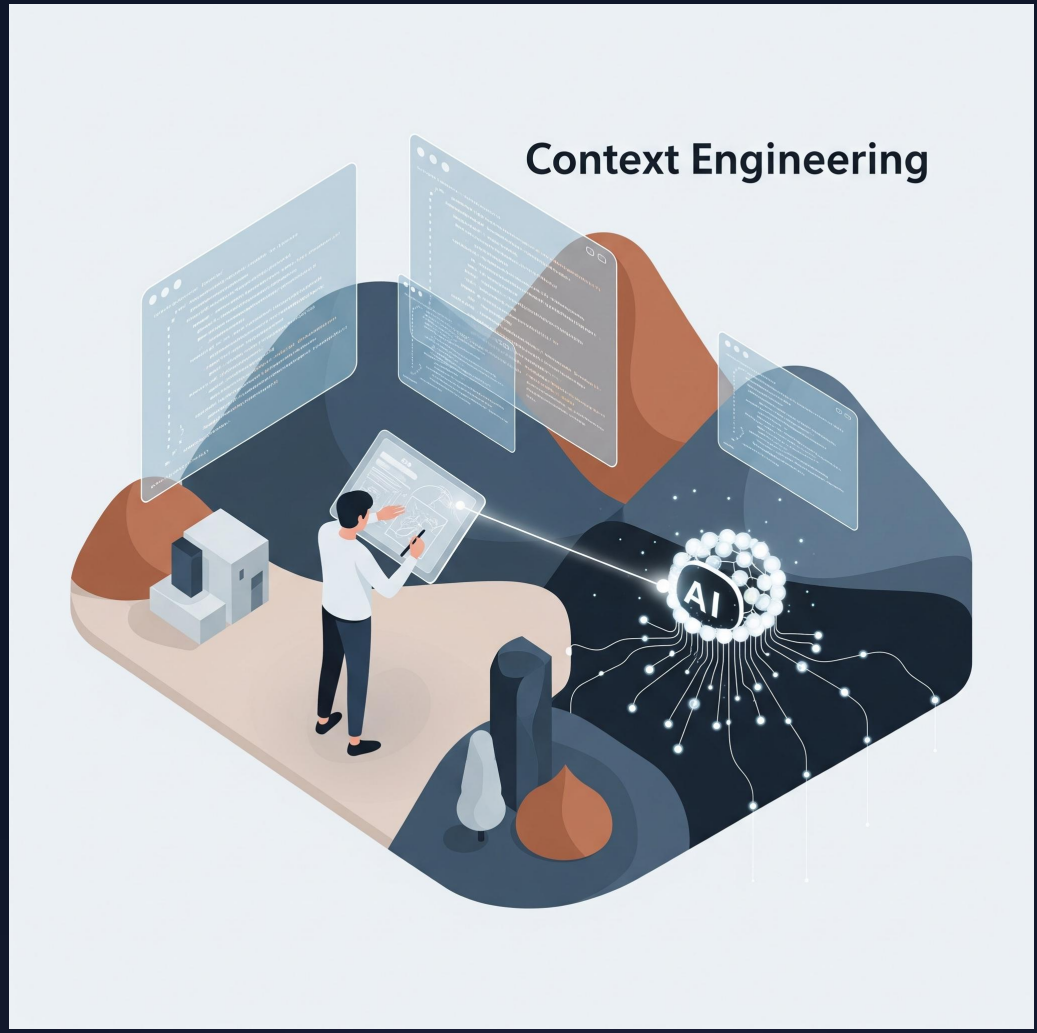
Mastering Context Windows, Memory Systems, and Steering with Claude Code & Gemini CLI

Beyond the Chatbot

This course bridges the gap between casual "prompt engineering" and rigorous **Context Engineering**.

We stop treating the context window as a chat box and start treating it as a **dynamic computing environment (RAM)**.

Learn to manage state, prevent "context rot," and ensure high-fidelity code generation for autonomous agents.



The Cognitive Architecture

Context as RAM

The context window is your agent's short-term memory. It is finite, expensive, and volatile. Treat it like a hardware resource.

Rigorous State

"Asking nicely" (prompting) fails at scale. We need engineering principles to manage instruction sets and memory.

Context Saturation

As the window fills with noise, reasoning degrades. This is "Context Rot." We must actively sanitize the environment.

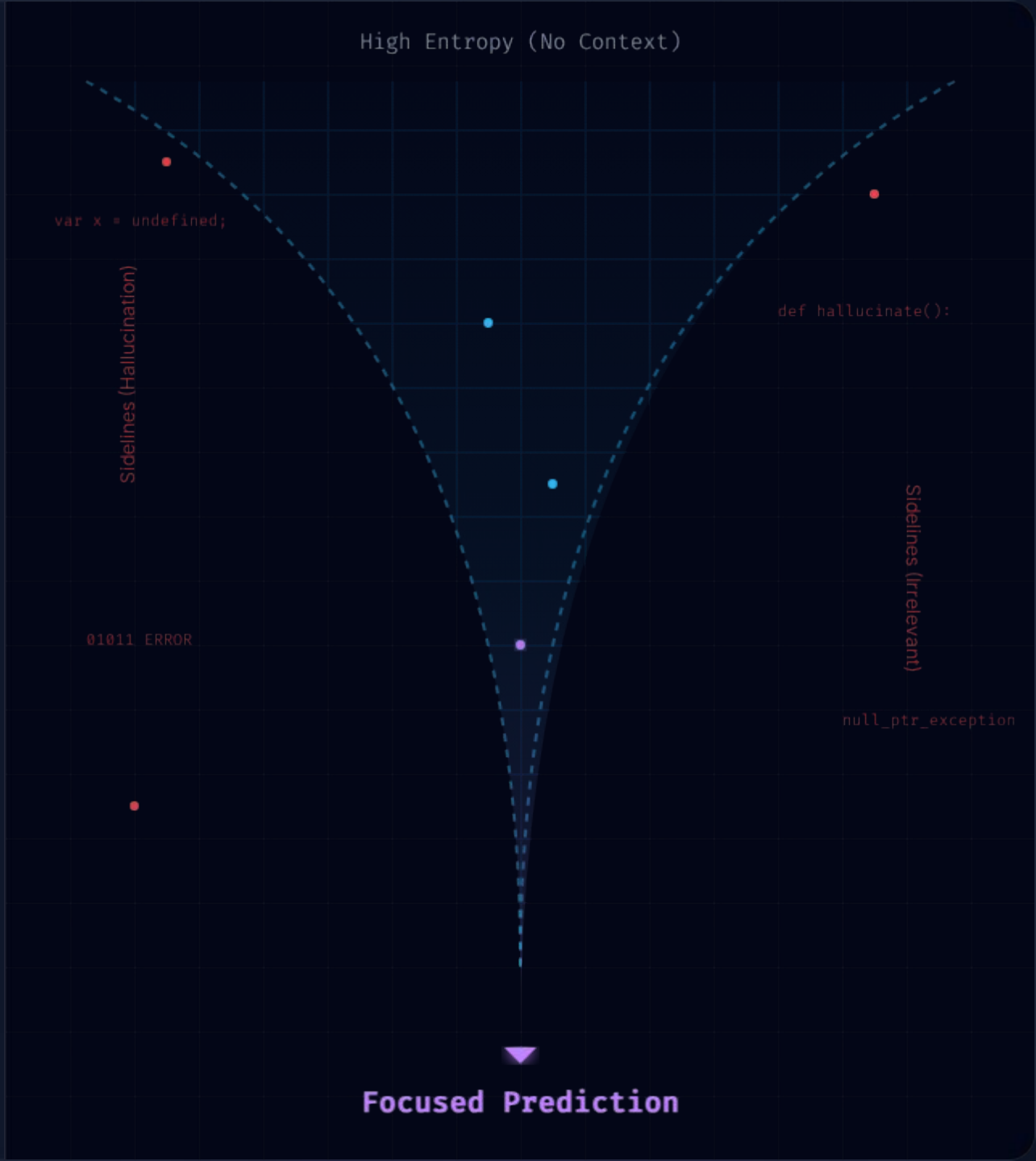
Vector Space Steering

The Probability Cone

Without context, the "next token" distribution is wide. The model risks drifting into the "sidelines" (hallucination).

The Focused Beam

High-quality context collapses the probability distribution, forcing the model to select from a narrow peak of relevant tokens.



Working Memory: The Hot Path

Definition: The immediate set of tokens the model is processing *right now* (Instructions + Recent History).

Recency Bias: The "Needle in a Haystack" problem means placement matters. Instructions at the very end (the hot path) are adhered to most strictly.

Breaking Point: We will stress-test the window to find exactly when instruction adherence fails.

The CLI Revolution

Claude Code

The Architect. Designed for the Plan → Act loop. It excels at multi-step reasoning and modifying its own behavior based on project structure.

Gemini CLI

The Library. A powerhouse for long-context tasks. With a 1M+ token window, it can ingest entire codebases without flinching.

Engineering Memory Systems

Static Context Injection

Using .md configuration files to permanently steer behavior. This moves "memory" out of the volatile context window and into the stable filesystem.

Context Caching

A strategy to reduce latency and cost. By identifying high-frequency, read-only data (like API docs), we cache it once and reference it cheaply.

Sanitation & Hygiene

The Compaction Strategy

Using commands like /compact or /compress to summarize conversation history. This retains the "thread" of logic while freeing up valuable tokens for new code generation.

Context Bankruptcy

Knowing when to use /clear. Sometimes, a context is so polluted with errors that the only engineering solution is to wipe the slate clean and re-inject the state.

Steering & Control Flow

Role-Based Context

Dynamically shifting the agent's persona. "You are a QA Engineer" yields different code than "You are a Prototyper."

The Scratchpad

Forcing blocks. This forces the model to "think out loud" and reason before committing to code.

Plan vs. Act

Architecting before coding. We will practice using "Plan Mode" to outline complex features before writing a single line.

Capstone: The Refactor

Step 1

Define Standards
CLAUDE.md

Step 2

Load Legacy Docs
(Context Caching)

Step 3

Plan & Refactor
(Steering Loop)

Step 4

Compact History
(Sanitation)

Questions?

Let's discuss the future of Agentic Workflows.