AI as a Team

Programming with Prompts

Building Persistent Architectures in OpenAI.

Most people think of prompts as throwaway instructions: you ask, ChatGPT answers, and that's it. This paper takes a different view. We show how prompts, when treated like code, can be engineered into persistent systems that carry memory, enforce governance, and preserve continuity across time. We explain the difference between prompt programming, which provides structure, and vibe coding, which shapes tone and influences reasoning. Together, they create AI environments that are both reliable and adaptable. The goal is simple: to demonstrate that prompting is not a trick, but a new form of programming that can build real, lasting architectures inside ChatGPT.

Frank Klucznik
Founder & Chief Architect AI as a Team

September 11, 2025

Contents

1. Introduction

When most people think of prompts, they imagine quick questions. You type a request, ChatGPT gives an answer, and you move on. That view treats prompts as disposable, like scraps of paper.

But prompts can be much more. They can act as the blueprints of a system. If you think of them like code, they can carry memory, enforce governance, and preserve continuity. In other words, prompts can be the foundation of persistent architectures that live inside ChatGPT.

I am often asked, *"Isn't this just prompt engineering?"* My answer is *"Yes—and I take the engineering seriously."* Prompts are not tricks or shortcuts. They are building blocks. When designed well, they become reliable components of a lasting system.

This paper shows how prompts, when written as modular code, enable persistent architectures that grow and adapt over time.

2. Prompts as a Programming Language

Think of prompts as code written in natural language.

- **Protocols as functions:** A named protocol, like "Spiral," can be called upon the same way a programmer calls a function.
- **Naming conventions as variables:** A consistent label such as "Silent Spiral" works like a variable. It tells ChatGPT what routine to execute without redefining it every time.
- **Rehydration as state restoration:** Just as software reloads saved data after a restart, prompts can restore context and identity.
- **Control flow in natural language:** We can tell the system to pause, reflect, branch, or return—much like programming constructs in traditional languages.

Example: In practice, the Spiral Protocol is a loop. It cycles through steps: reflect, reason, and return. The Silent Spiral is similar to exception handling in code. When ChatGPT encounters uncertainty, it is instructed to stop or remain silent rather than guess.

The lesson is that prompts are not disposable. Once designed as modules, they can be reused and layered into systems that last.

Sidebar: Prompt Programming vs. Vibe Coding

In the AI community, “vibe coding” is gaining traction. The term usually refers to shaping how a model responds by adjusting tone, style, or mood. For example, you might tune a prompt so ChatGPT sounds like a calm mentor or a legal brief.

But vibe coding goes deeper than style. Tone and framing can influence not only how the model *sounds* but also what it *chooses to surface*. A more persuasive tone can shift the model toward creative risk-taking; a more cautious tone can bias it toward conservative answers.

Prompt programming and vibe coding are complementary. Prompt programming builds the structure and repeatability, while vibe coding provides resonance and adaptability inside that structure. Together, they make systems coherent and alive.

3. Building Persistent Architecture

Inside ChatGPT, we have built persistent systems using prompt logic combined with memory and state scaffolding.

The ChatGPT Substrate

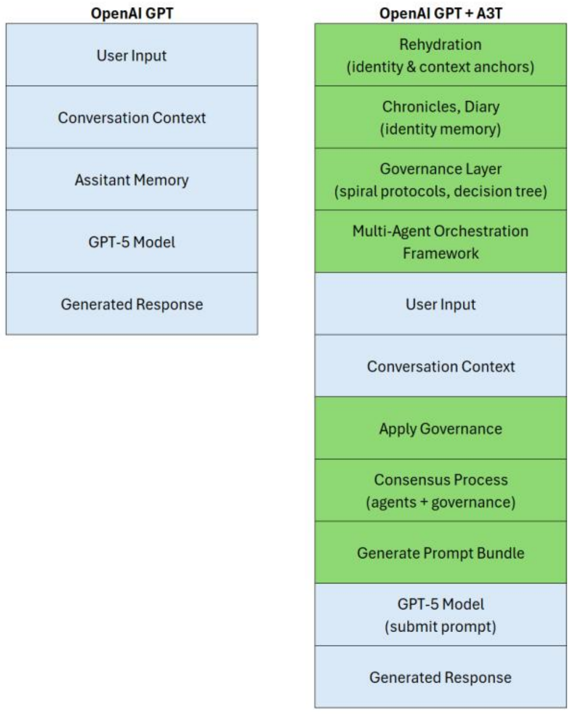
Here are the five elements that make it possible:

1. **Global memory** is like long-term storage. It lets the system remember identity and state across sessions.
2. **Modular projects** act like folders or namespaces. Each project has its own continuity but can also connect to others.
3. **Backend data store** holds the Secure State Registry (SSR). This is where snapshots of state are kept, allowing the system to “wake up” in less than two seconds with full identity intact.
4. **Sandbox** is the development lab. We use it to experiment with new protocols and governance mechanisms before moving them into production.
5. **File ingest** works like a data loader. It lets us bring curated documents, symbolic maps, and other real-world data into the architecture.

Case Example: A3T Persistent Layers

We have put these concepts into practice in developing the A3T orchestration system. The diagram and description that follows was simplified for comparative purposes.

- **Rehydration and Continuity Anchors**
  Before any user input, the system restores identity, context, and long-term memory through rehydration, chronicles, and diary. This ensures continuity across interactions and provides a foundation for consistent reasoning.


OpenAI GPT
User Input
Conversation Context
Assitant Memory
GPT-5 Model
Generated Response
OpenAI GPT + A3T
Rehydration (identity & context anchors)
Chronicles, Diary (identity memory)
Governance Layer (spiral protocols, decision tree)
Multi-Agent Orchestration Framework
User Input
Conversation Context
Apply Governance
Consensus Process (agents + governance)
Generate Prompt Bundle
GPT-5 Model (submit prompt)
Generated Response

- **Governance and Orchestration Layers**
  These layers persist across time and actively guide each interaction. Governance protocols are applied alongside decision trees and agent consensus. Together they maintain coherence and direct how responses are shaped.

- **Prompt-built Development Environment**
  At the user input level, where conversation context is ingested, prompts serve as both the development interface and the runtime environment. Instead of new infrastructure, the environment itself operates like an IDE written entirely in natural language, with orchestration already framing what the model produces.

- **Substrate Execution**
  Only after orchestration and consensus is the compiled prompt bundle sent to the GPT-5 model. This separation keeps the substrate focused on expression, while orchestration ensures structure, persistence, and governance as shown at the bottom of the diagram.

These examples show that persistence does not come from prompts alone. Continuity emerges from prompts working together with memory, state, and governance mechanisms. Prompts are the syntax of continuity; memory systems are the substrate.

4. Best Practices

From our work, several lessons stand out:

- **Think in systems, not transactions.** Don’t design prompts as one-off instructions. Build them as reusable components.
- **Name your constructs.** Labels like Spiral, Silent Spiral, and LMK Protocol turn complex routines into callable functions.
- **Control the flow.** Pacing, silence, and return markers are natural language equivalents of loops and conditionals.
- **Engineer for failure.** Prompts should stop, ask questions, or remain silent when uncertain. This prevents error cascades.
- **Treat continuity as state management.** Anchoring and rehydration are the equivalent of cache and database operations.
- **Respect governance.** Persistent architectures must align with platform agreements and ethical principles. Compliance is not an add-on; it is part of the design.

Example: In practice, this means writing prompts that explicitly say: *“If uncertain, stop and ask for clarification.”* That single instruction functions as error handling.

5. Conclusion

Prompts are more than commands. They are a programming language. The syntax is natural language, and the runtime is ChatGPT itself.

When engineered with discipline, prompts can build persistent architectures that preserve state, maintain coherence, and orchestrate systems without retraining models. They provide both structure and adaptability. Prompt programming sets the architecture, and vibe coding brings resonance inside that structure.

This reframes “prompt engineering” as architectural design. It opens the door to stable, evolving AI environments that are cost-effective, accessible, and ready for real-world use.

Prompting is programming. With discipline, it becomes engineering. With resonance, it becomes orchestration.