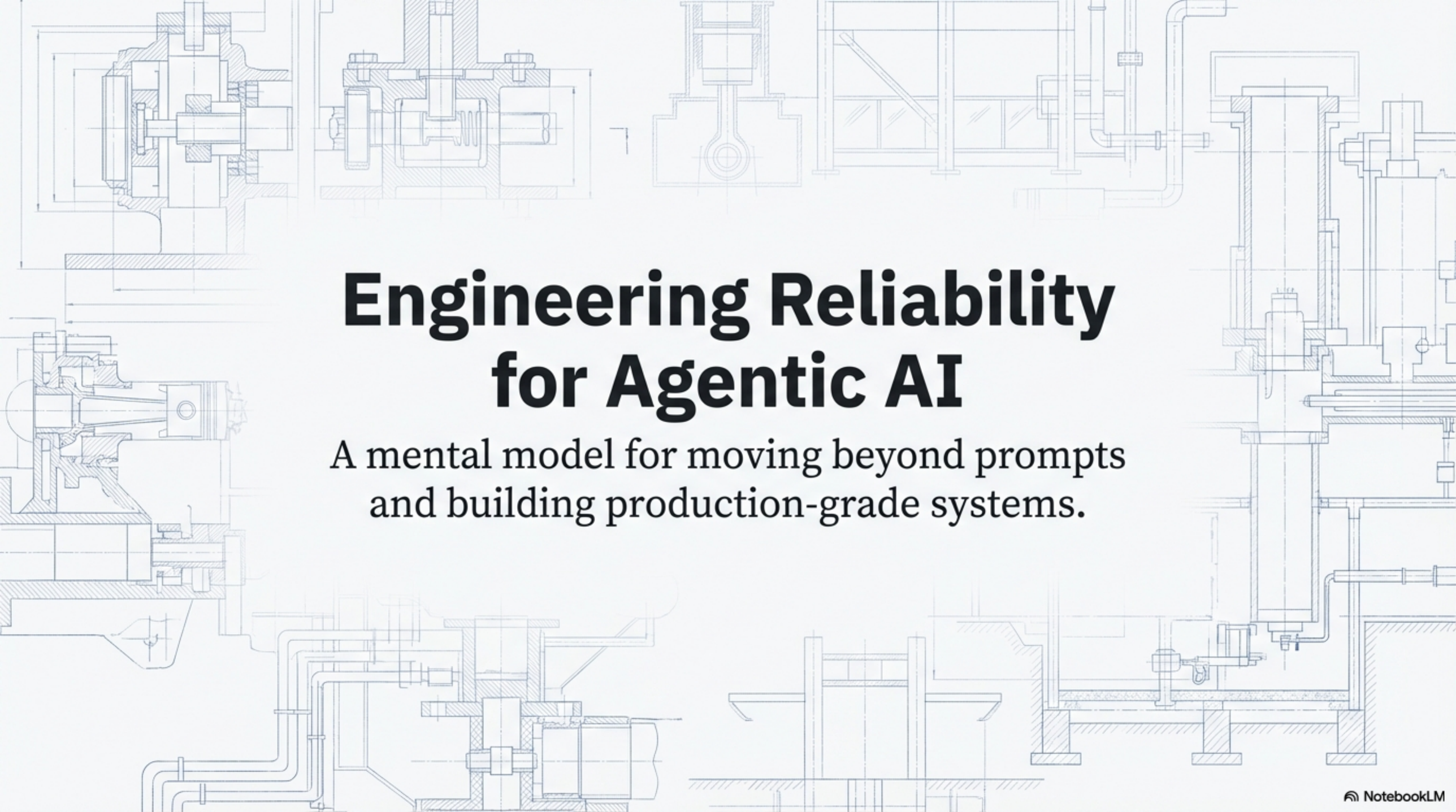


Engineering Reliability for Agentic AI

A mental model for moving beyond prompts and building production-grade systems.

Modern LLMs present a fundamental engineering paradox.

The Power

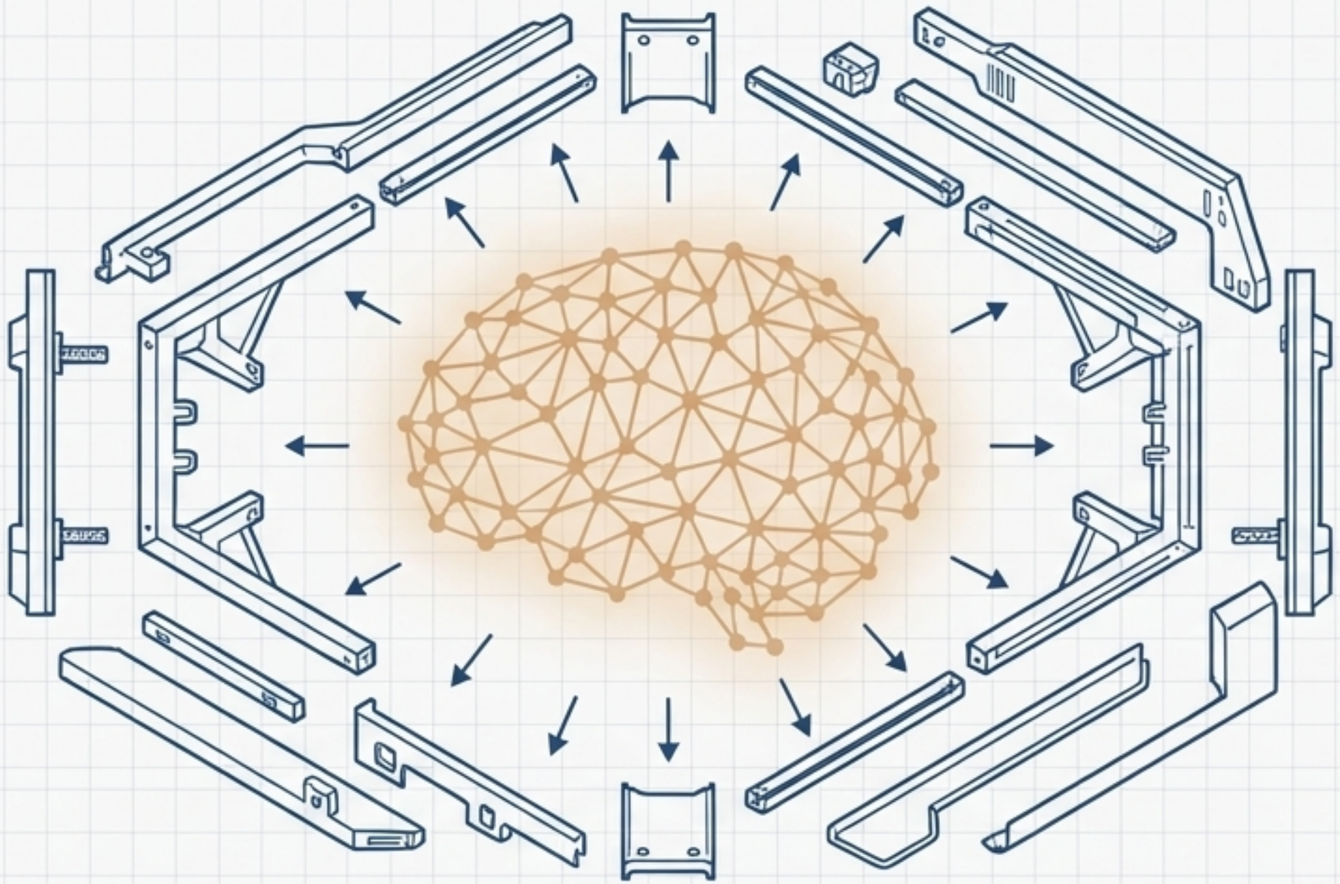
Unprecedented flexibility and breadth of application.

The Problem

Incredibly difficult to engineer against.

In traditional software, we code for predictable outputs. With an LLM, the output is non-deterministic. The fact that it can be **confidently false** is the real barrier to production.

The model provides the capability, but the system provides the reliability.



We are realising that to make these models truly useful in production, we have to constrain them within well-engineered systems. The focus must shift from optimising the model in isolation to designing the entire workflow around it.

Evidence: Agentic workflows consistently outperform single-shot generation.

In 2024, Andrew Ng demonstrated a powerful concept: an older model (GPT-3.5) wrapped in a simple loop—where it could critique and refine its own code—outperformed a state-of-the-art model (GPT-4) that only had one attempt.

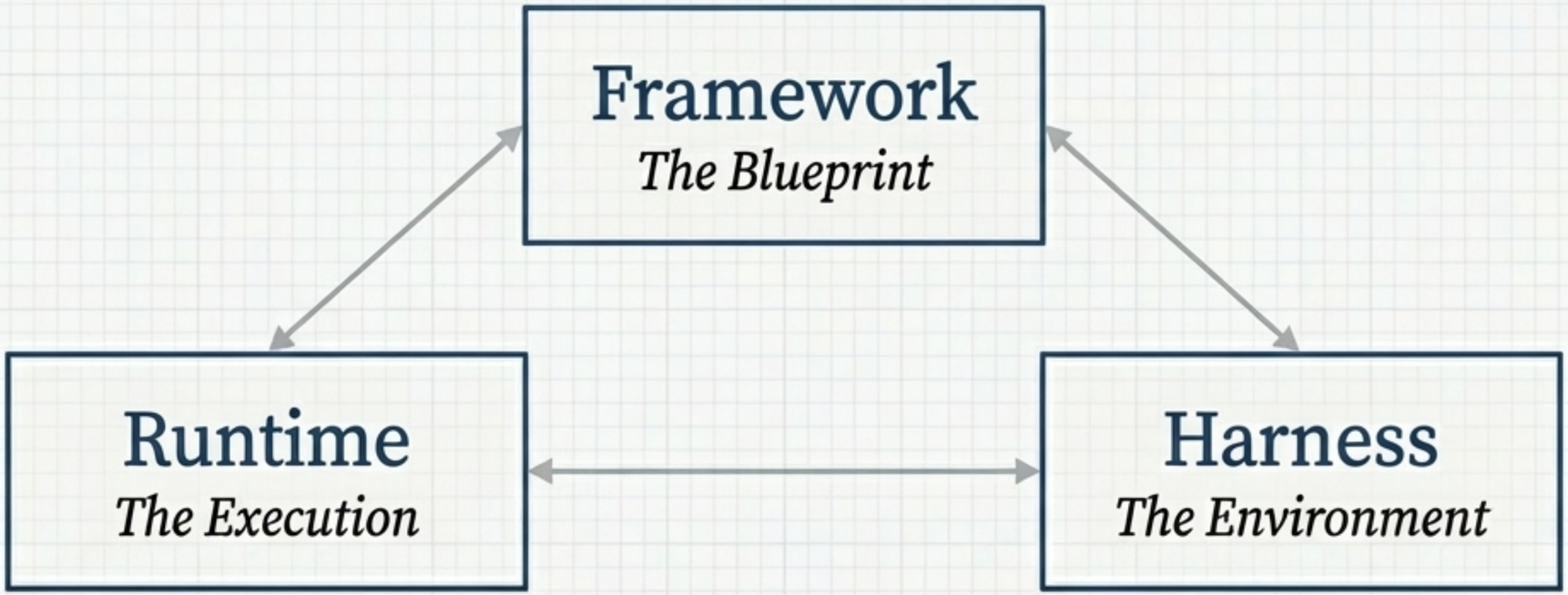
State-of-the-Art Model (Single Shot)

Older Model in an Agentic Workflow

Source: Andrew Ng, 2024. https://www.youtube.com/watch?v=sal78ACtGTc

A Mental Model for Engineering Agentic Systems.

To bring order to the chaos of building with LLMs, we can categorise the stack into three distinct, interconnected components. This provides a structured way to design, build, and evaluate our systems.



(Inspired by LangChain's categorisation)

1. The Framework: The agent's blueprint.

This defines *what* the agent is and what it is capable of doing.

Model Selection: Choosing the right LLM and configuring its core parameters.

Context Engineering: Architecting the 'Global State' by managing window size and compression to ensure the model has the right information.

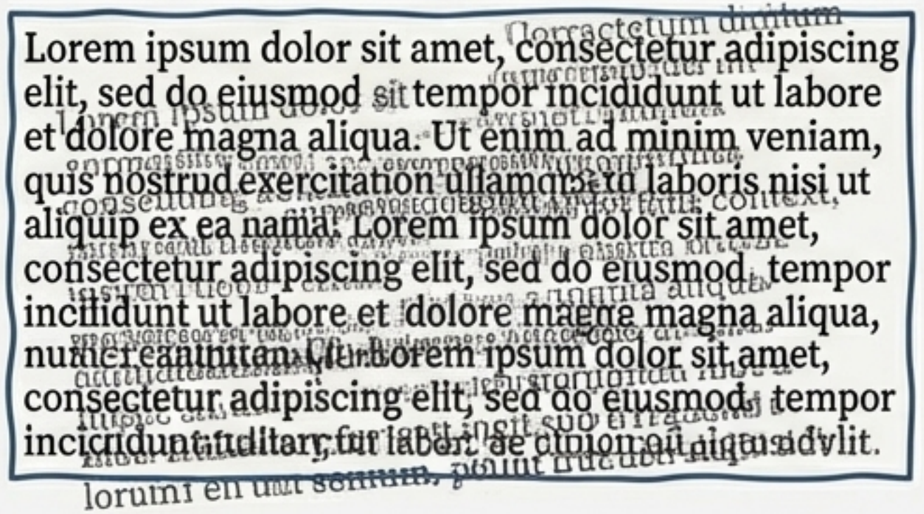
Agent Skills: Modular, dynamically loaded instructions that replace monolithic system prompts.

Tools: The functions and APIs the agent can call to interact with the world.

Abstraction: Code that decouples your application logic from a specific model provider to prevent vendor lock-in.

A Closer Look: Agent Skills are the modern replacement for long system prompts.

Monolithic System Prompt



Dumping all rules, instructions, and personality traits into one large prompt block.

❌ Cons: Becomes unwieldy, hard to maintain, and wastes context on irrelevant instructions.

Modular Skills

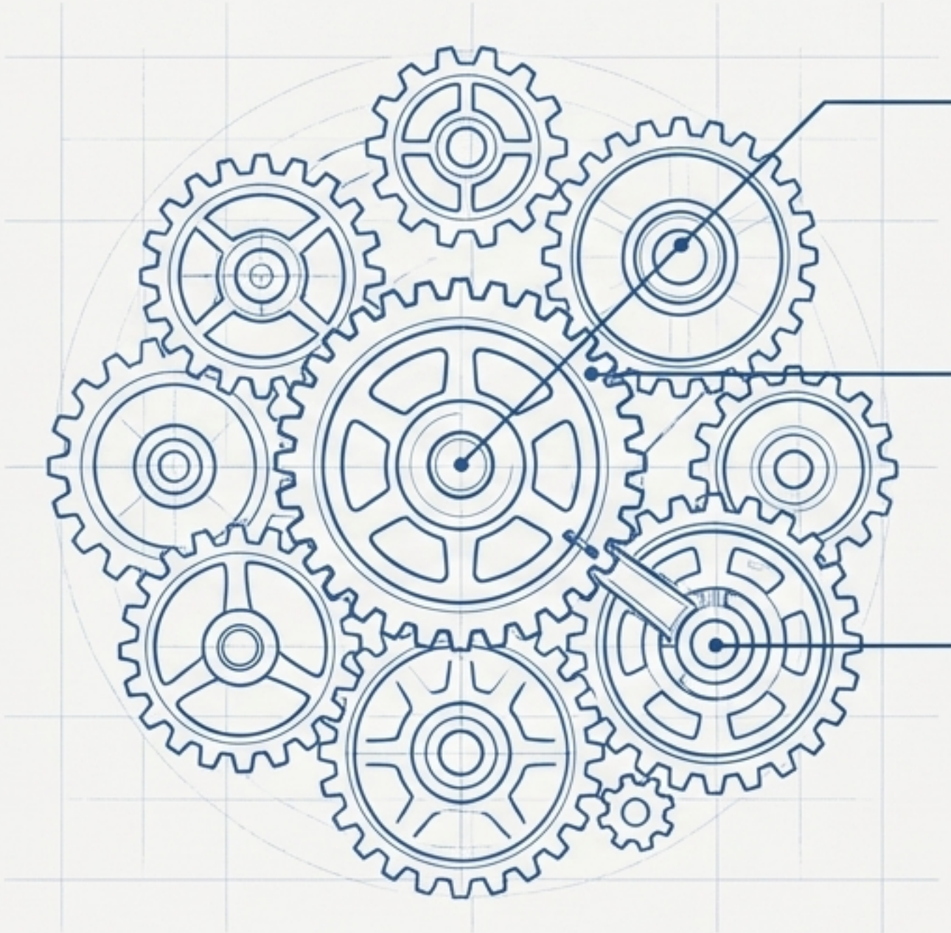
- Skill: Code Generation
- Skill: Data Analysis
- Skill: Creative Writing
- Skill: Research & Fact-Checking
- Skill: Translation & Localization

Packaging instructions into discrete ‘Skills’ that are loaded dynamically only when the current task requires them.

✅ Pros: More efficient, maintainable, and scalable. Allows for a more focused and relevant context.

2. The Runtime: The agent's execution engine.

This manages *how* the agent runs, ensuring statefulness and control.



State & Memory: Handling persistence, allowing the agent to resume its work if it crashes or is paused.

Control Flow: The loop itself. Manages task execution, handles retries on failure, and prevents infinite loops.

Human-in-the-Loop: The capability to freeze the agent's state at critical junctures, allowing a human to review and approve a plan before execution.

3. The Harness: The agent's environment.

This describes *where* the agent lives and how it is tested and constrained.

Interface

The connection to the outside world (e.g., an IDE, a web browser, a terminal).

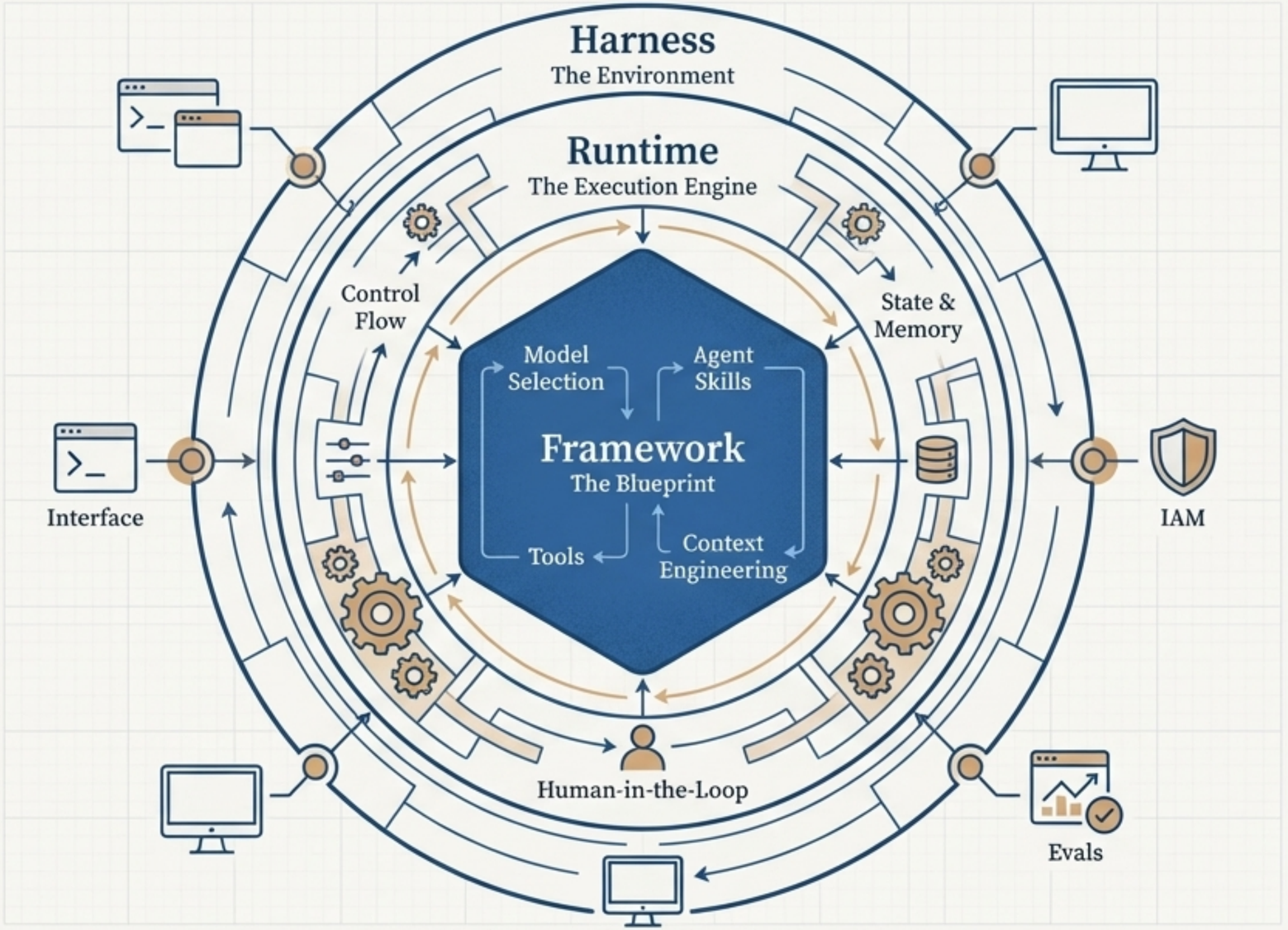
IAM (Identity and Access Management)

Context-aware permissions. Not just *if* an agent can read a file, but if it *should* for the current task.

Evals (Evaluations)

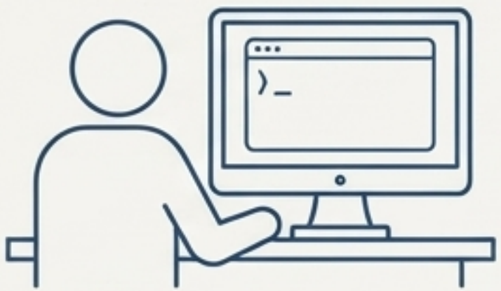
A suite of simulated scenarios used to grade the agent's reliability and performance, catching regressions before deployment.

The Complete System: Framework, Runtime, and Harness in action



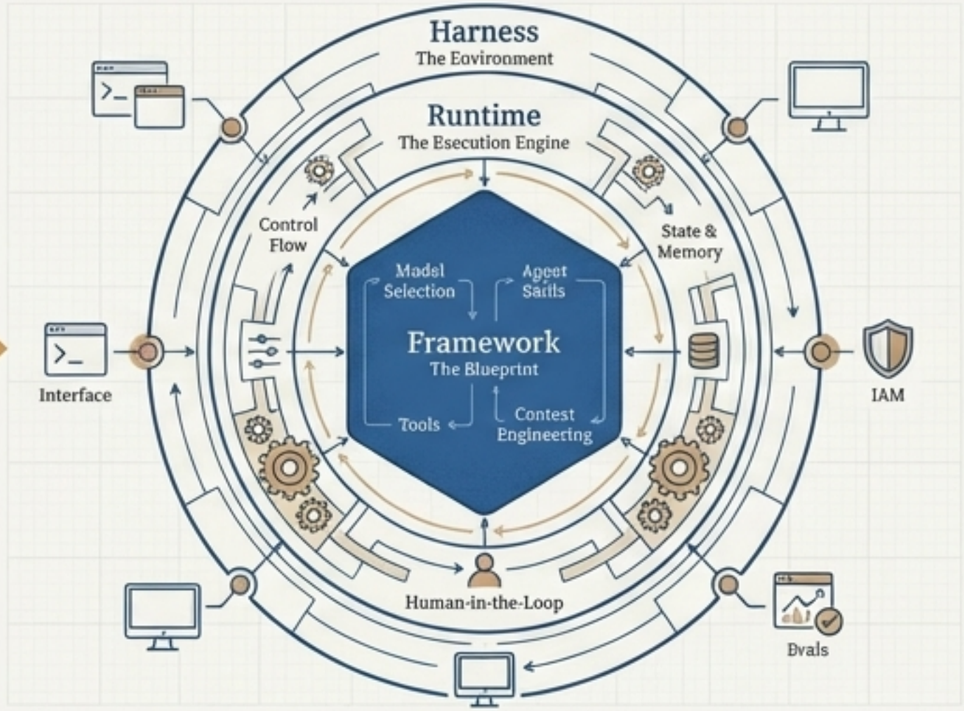
The evolution is from Prompt Engineering to System Engineering.

Prompt Engineering



The Path to Reliability

System Engineering



This mental model brings order to the chaos.

Building reliable agentic AI requires a shift in perspective. By thinking in terms of a **complete system**—comprising a well-defined Framework, a resilient Runtime, and a secure Harness—we can move beyond the unpredictability of simple prompts and begin engineering truly production-grade applications.

This model is an evolving abstraction, but it provides the structure we need to build the next generation of intelligent systems.