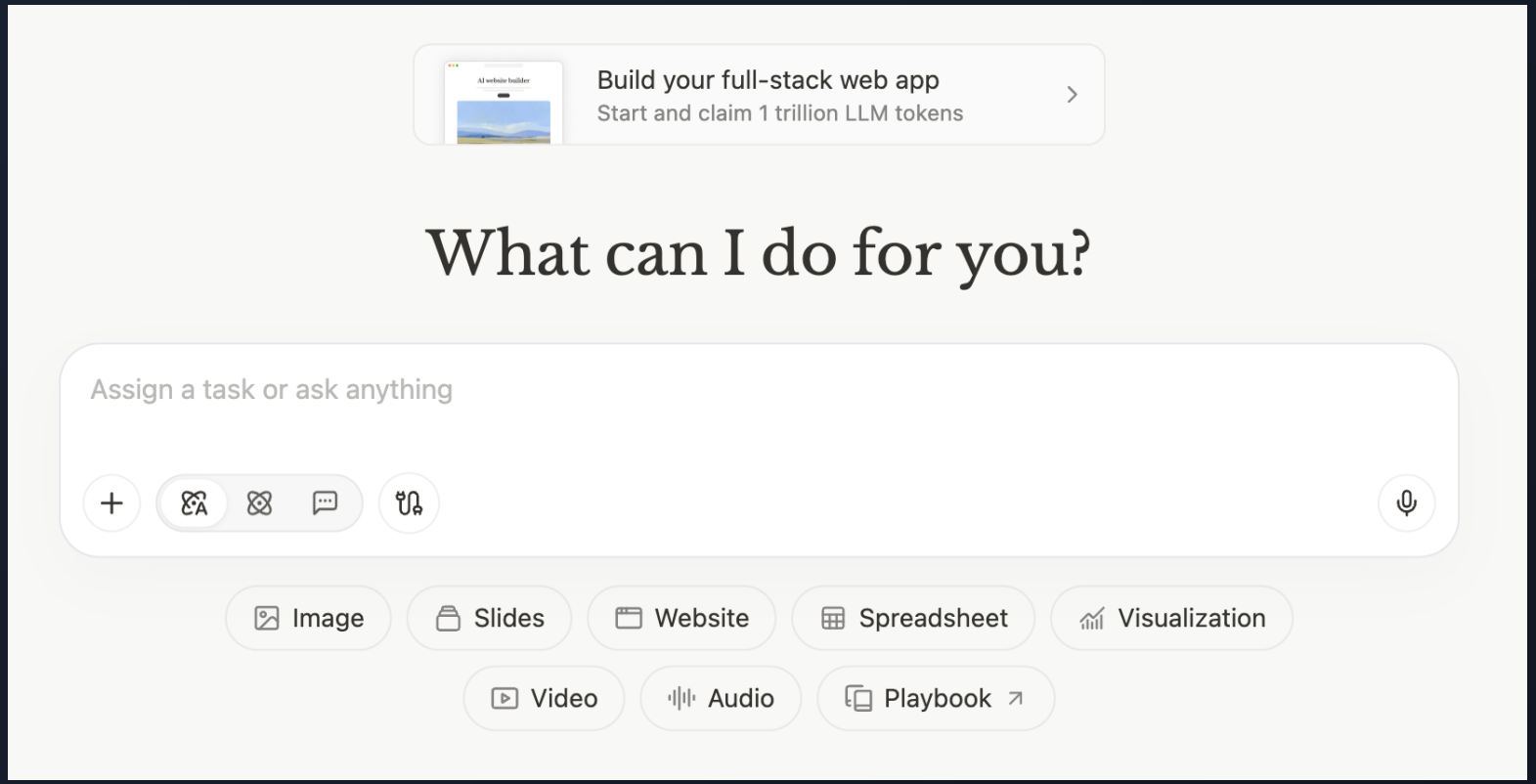
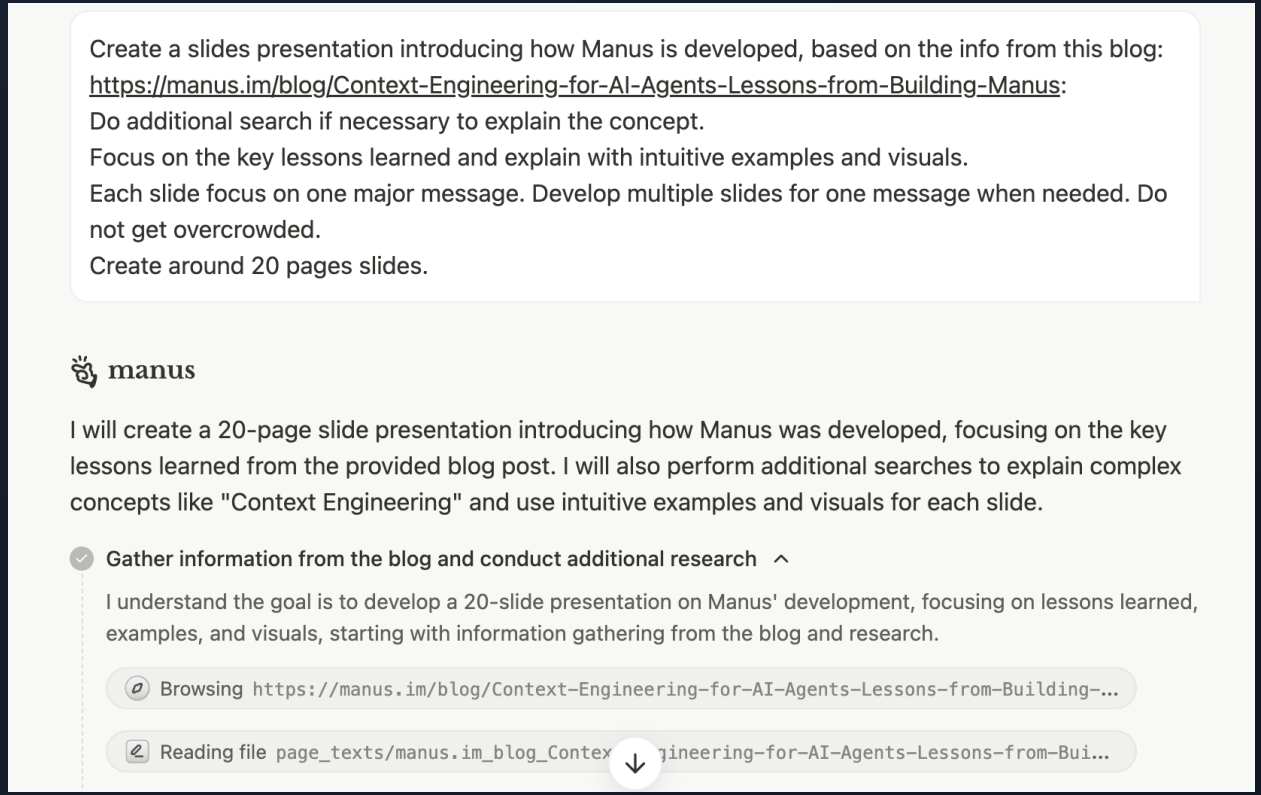
Building Effective AI Agents for Millions of Users

Lessons from Building Manus

Who is Manus: manus.im



I used Manus to create this presentation!



The Learning Journey

Manus learned these lessons through **repeated rebuilds** and thousands of hours of **real-world testing** across diverse tasks.

Training vs. Context: A Critical Decision at the Start

At the beginning of Manus, the team faced a pivotal choice **train** an end-to-end agentic model from scratch, or **build on top** of frontier models' in-context learning abilities.

In the BERT era, fine-tuning took **weeks per iteration.** For fast-moving applications, especially pre-PMF (product market fit) products, such slow feedback loops are deal-breakers.

Hard-earned lesson: Previous startup trained models for information extraction and semantic search. Then GPT-3 launched, and those models became irrelevant overnight.

2018 — **BERT Era**

Models required fine-tuning and evaluation before transfer to new tasks. Weeks per iteration, even for tiny models.

2020 — **GPT-3 & Flan-T5**

In-house trained models became irrelevant overnight. These models marked the beginning of in-context learning.

2025 — **Manus Decision**

Choose context engineering over training. Ship improvements in hours instead of weeks.

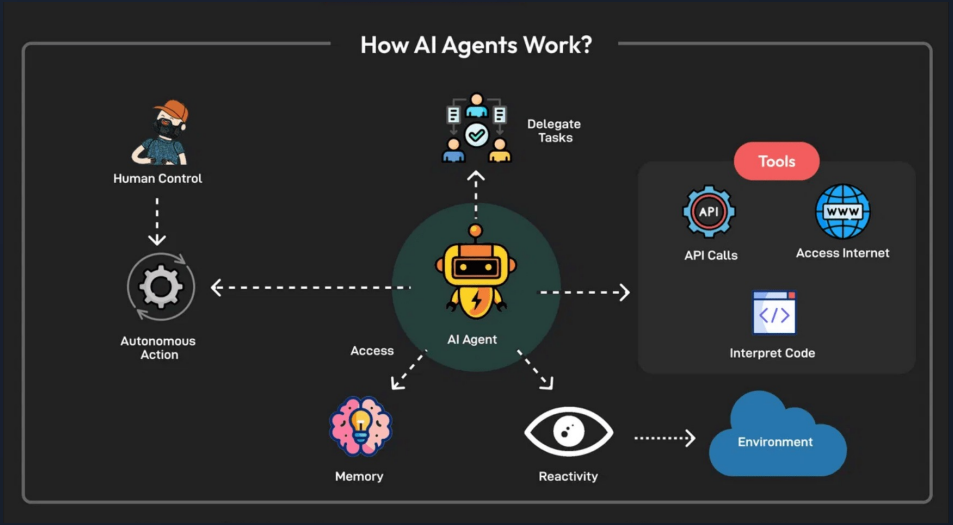
Context Engineering Enables Hours, Not Weeks, of Iteration

Manus chose to **bet on context engineering,** allowing rapid iteration and keeping the product orthogonal to underlying model improvements.

> ***"If model progress is the rising tide, we want Manus to be the boat, not the pillar stuck to the seabed."***

- **Ship improvements in hours:** instead of weeks through prompt engineering
- **Stay orthogonal to models:** As frontier models improve, Manus improves automatically
- **Flexibility and adaptability:** No need to retrain when new capabilities emerge

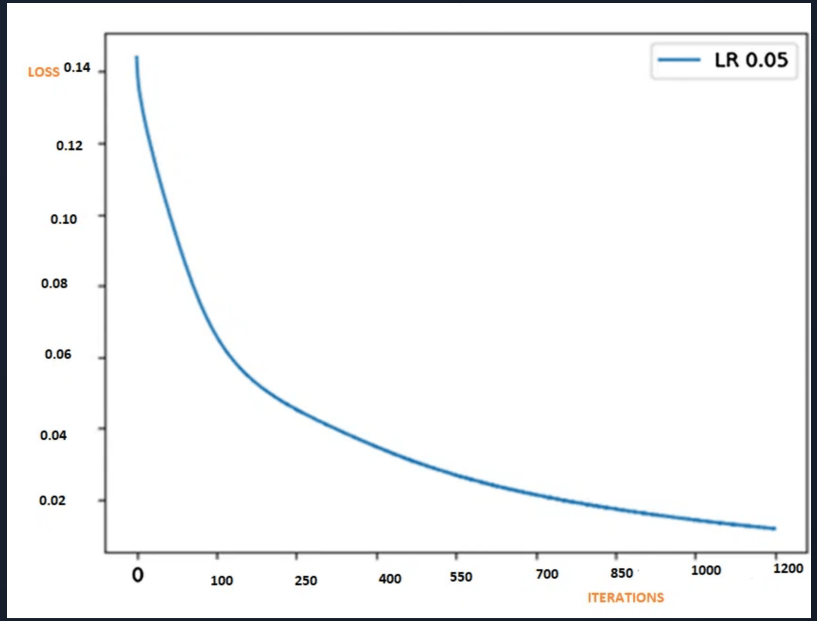
How AI agents work: The bet on Context Engineering



1. **User provides input** to start the task
2. **Agent selects action** from predefined action space based on current context
3. **Action is executed** in environment (e.g., Manus's VM sandbox)
4. **Execution produces observation** with results or feedback
5. **Action and observation appended** to context
6. Loop continues until task is complete

Average Manus task: ~50 tool calls

Four Complete Rebuilds Through "Stochastic Gradient Descent"



Context engineering turned out to be anything but straightforward. Manus **rebuilt its agent framework four times,** each time after discovering a better way to shape context.

The "SGD" Process

1. Architecture searching
2. Prompt fiddling
3. Empirical guesswork
4. Iterate until convergence

> ***"It's not elegant, but it works."***

The presentation shares the **local optima** arrived at through this manual optimization process.

Six Principles for Building Production AI Agents

Through extensive experimentation, Manus discovered six critical principles that define effective context engineering

01

Design Around the KV-Cache

KV-cache hit rate is the single most important metric, directly affecting both latency and cost

02

Mask, Don't Remove

Use state machines to control tool availability without modifying context definitions

03

Use the File System as Context

Treat file system as unlimited, persistent, externalized memory for the agent

04

Manipulate Attention Through Recitation

Recite objectives into end of context to avoid lost-in-the-middle problems

05

Keep the Errors

Leave failed actions in context so the model can learn from errors and adapt

06

Don't Get Few-Shotted

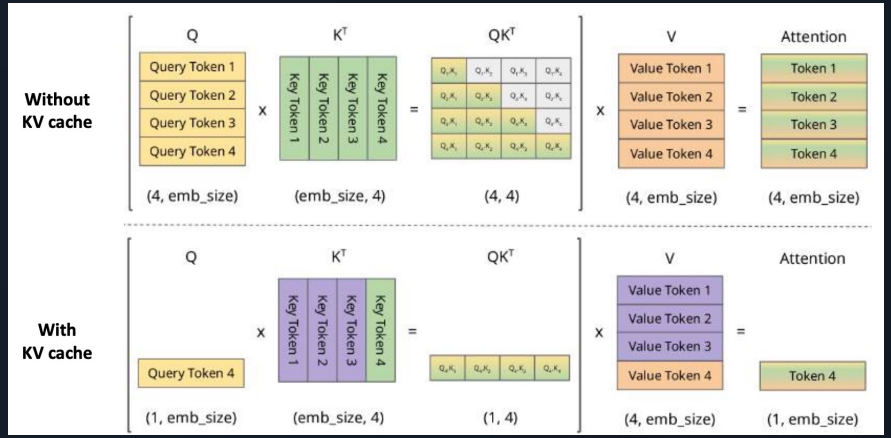
Introduce controlled variation to prevent pattern mimicry and brittle behavior

Lesson 1

KV-Cache Hit Rate: The Single Most Important Metric

If you could choose only one metric for a production AI agent, KV-cache hit rate directly determines both latency and cost.

KV-cache stores intermediate **key-value computations** from the attention mechanism, allowing reuse of calculations for identical context prefixes.

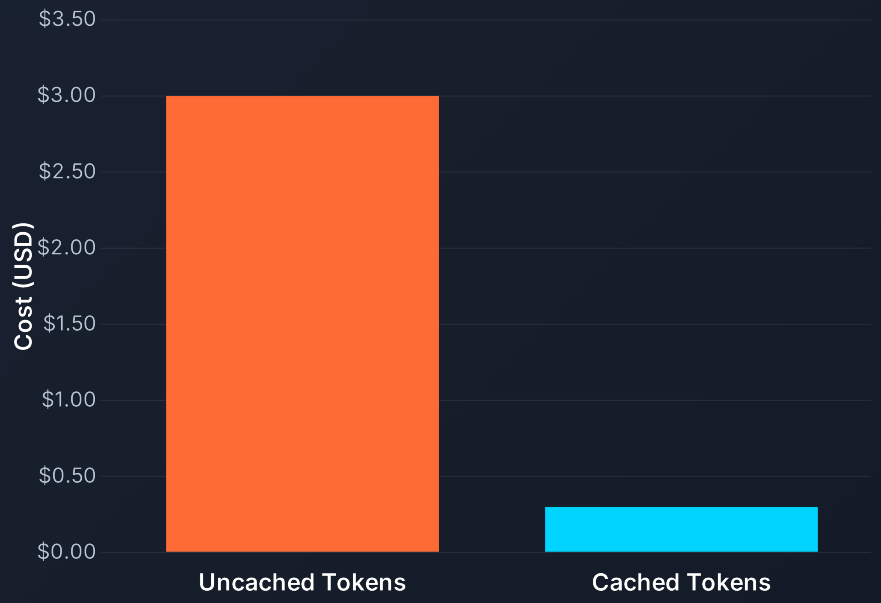


Values that will be taken from cache

100:1
Input-to-output token ratio in Manus agents

~50
Average tool calls per Manus task

10x Cost Difference Between Cached and Uncached Tokens



Source: Claude Sonnet API pricing (2025)

Cost Savings

10x

Cached tokens cost **$0.30/MTok** vs uncached at **$3.00/MTok**

Speed Improvement

5.21x

Faster generation with KV-caching enabled

Production Impact

Massive

For agents processing millions of tokens daily, savings compound dramatically

KV-Cache Best Practice #1

Keep Your Prompt Prefix Stable — Every Token Matters

Due to the **autoregressive nature** of LLMs, even a single-token difference invalidates the cache from that point onward. Every token depends on all previous tokens. When one token changes, all subsequent computations must be recalculated.

✓ Good: Stable Prefix

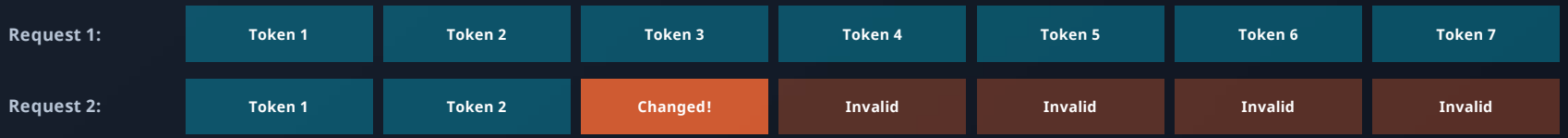
System: You are a helpful AI assistant...
[Static prompt - cache reusable]

User: What time is it?
[Dynamic info comes later]

✗ Bad: Changes Every Request

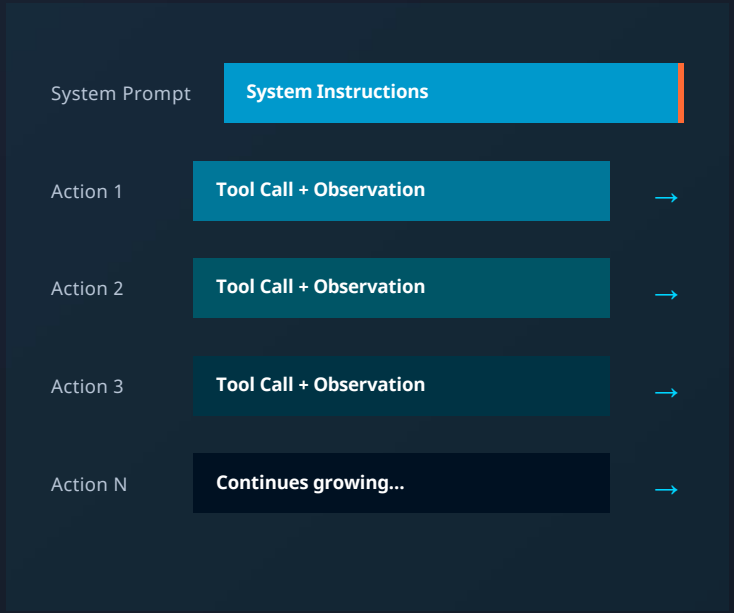
System: Current time is 2025-10-24 11:23:47
You are a helpful AI assistant...
[Cache invalidated every second!]

Cache Invalidation Cascade Effect



KV-Cache Best Practice #2

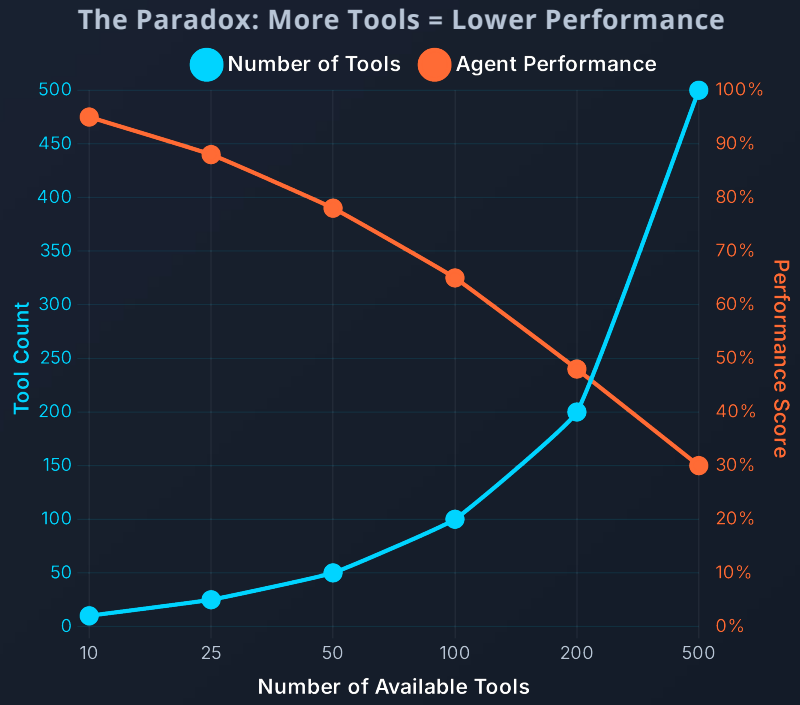
Append-Only Context and Explicit Cache Breakpoints



Never modify previous actions or observations.
Context should only grow forward.

Lesson 2

More Tools Make Agents Dumber, Not Smarter



As your agent takes on more capabilities, its action space naturally grows more complex. The **Model Context Protocol (MCP)** only adds fuel to the fire—users can plug in hundreds of tools.

The Result

- Model more likely to select **wrong action**
- Agent takes **inefficient paths**
- Your heavily armed agent gets **dumber**

The intuitive reaction—dynamically loading tools on demand—breaks KV-cache and confuses the model.

Context-Aware State Machine Controls Tool Availability

✗ Bad Approach

Dynamically Remove Tools

Loading tools on demand using RAG-like approaches seems intuitive but creates serious problems:

- **Invalidates KV-cache:** for all subsequent actions and observations
- **Confuses the model:** when previous observations reference missing tools
- Leads to **schema violations** and hallucinated actions

✓ Manus Solution

Mask Token Logits

Rather than removing tools, Manus uses a smarter approach:

- **Keep all tool definitions:** in context to preserve KV-cache
- **Use state machine:** to determine which tools are available
- **Mask token logits:** during decoding to control action selection

The key principle: Mask, don't remove. Control behavior through constrained decoding, not context modification.

Context-Aware State Machine Controls Tool Availability

State Machine Example: Tool Availability by Context

Agent State	Available Tool Prefixes	Example Scenario
User Input Received	`message_*`	Must reply to user, not take action
Web Research Mode	`browser_*, search_*`	Only web-related tools available
Code Execution Mode	`shell_*, file_*`	Only command-line and file tools
General Task Mode	`All tools`	Full action space available

Tool name prefixes (e.g. `browser_*, shell_*`) enable easy enforcement of tool groups without **modifying context or breaking KV-cache**

Constrained Decoding Through Response Prefill

Three modes of function calling that constrain action space without modifying tool definitions

Mode 1

Auto

Model **may or may not** call a function. Gives the model freedom to respond directly or use a tool.

Prefill Implementation:

```
<|im_start|>assistant
```

Mode 2

Required

Model **must call a function**, but the choice is unconstrained. Any tool from the full action space.

Prefill Implementation:

```
<|im_start|>assistant<tool_call>
```

Mode 3

Specified

Model **must call a function from specific subset**. Enforces selection from particular tool group.

Prefill Implementation:

```
<|im_start|>assistant<tool_call>{"name": "browser_
```

Manus Example

When user provides new input, Manus must **reply immediately** instead of taking an action → use **Auto mode** constraint to force direct response

Lesson 3

Treat the File System as Unlimited, Persistent Context

Modern frontier LLMs offer context windows of **128K tokens or more.** But in real-world agentic scenarios, that's often not enough.

1 **Observations Can Be Huge**

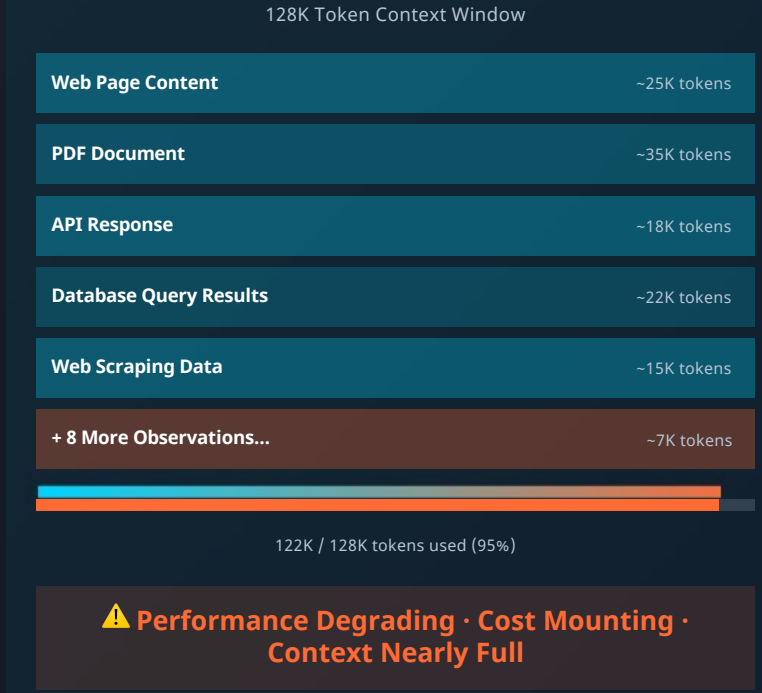
Web pages, PDFs, and unstructured data easily blow past context limits

2 **Performance Degrades**

Model performance tends to degrade beyond certain context length, even if window technically supports it

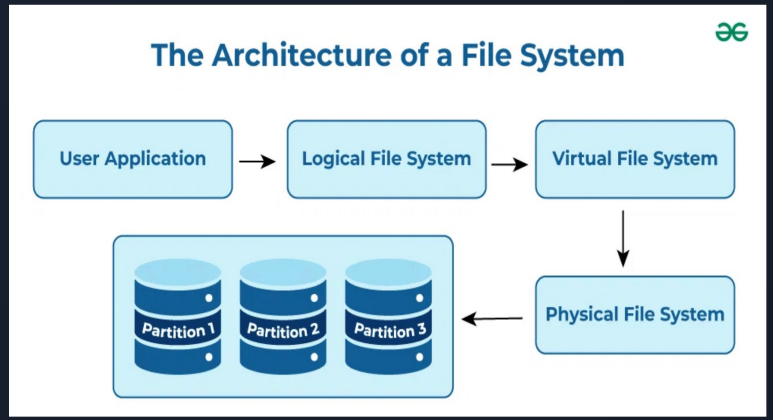
3 **Long Inputs Are Expensive**

Even with prefix caching, you're still paying to transmit and prefill every token



Fundamental problem: Can't predict which observation becomes critical 10 steps later. **Any irreversible compression carries risk.**

Treat the File System as Unlimited, Persistent Context



Manus uses the file system as **externalized memory:** the model learns to write to and read from files on demand, using it as structured, operable storage.

Three Key Properties

- ∞ Unlimited in Size
- 💾 Persistent by Nature
- ⚡ Directly Operable

Compression strategies are **always restorable:** shrink context length without permanently losing information.

Restorable Compression Examples

- Web page content can be dropped if `URL` is preserved
- Document contents can be omitted if `file path` remains available
- Agent can re-read files when needed for specific operations

Lesson 4

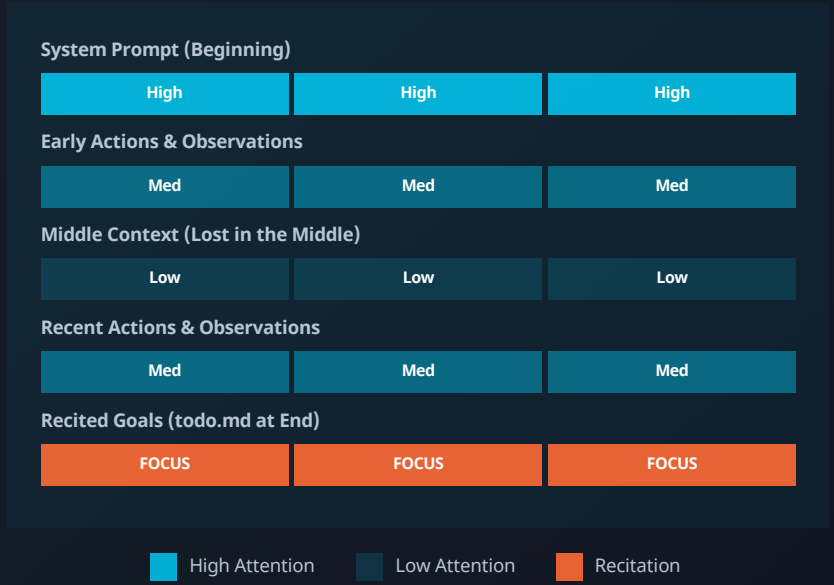
Use todo.md to manipulate the attention

Observation

When Manus creates and updates `todo.md` files step-by-step, checking off items as it progresses, it's deliberately manipulating the attention mechanism.

- Average task requires **~50 tool calls** —a long loop vulnerable to drifting off-topic or forgetting earlier goals, especially in long contexts.
- The **"lost-in-the-middle"** problem: models perform best with information at the beginning or end of context, but degrade in the middle.

Attention Pattern: U-Shaped Distribution



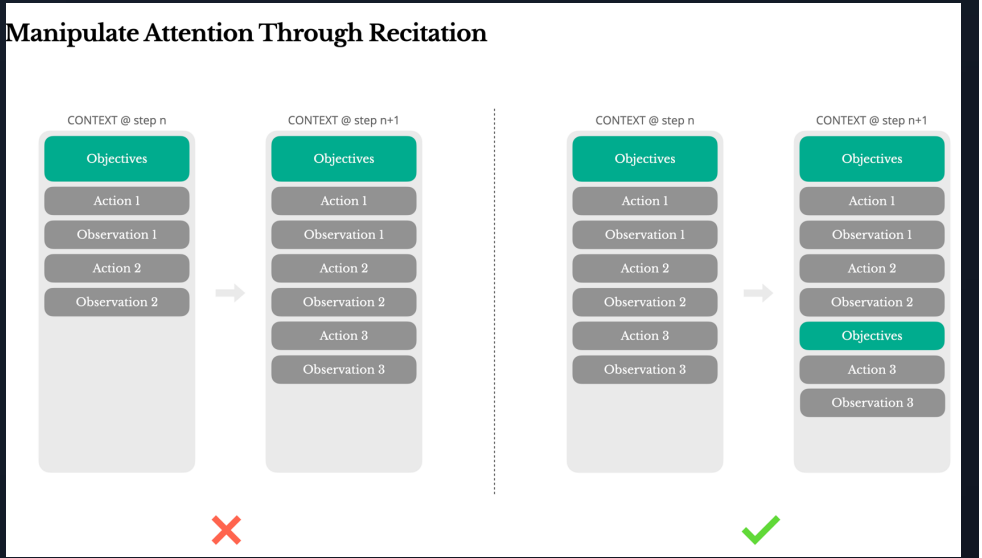
Lesson 4

Use todo.md to manipulate the attention

Observation

When Manus creates and updates `todo.md` files step-by-step, checking off items as it progresses, it's deliberately manipulating the attention mechanism.

- Average task requires **~50 tool calls** —a long loop vulnerable to drifting off-topic or forgetting earlier goals, especially in long contexts.
- The **"lost-in-the-middle"** problem: models perform best with information at the beginning or end of context, but degrade in the middle.
- Manus **recites objectives into the end of context** , pushing the global plan into the model's recent attention span.



Lesson 5

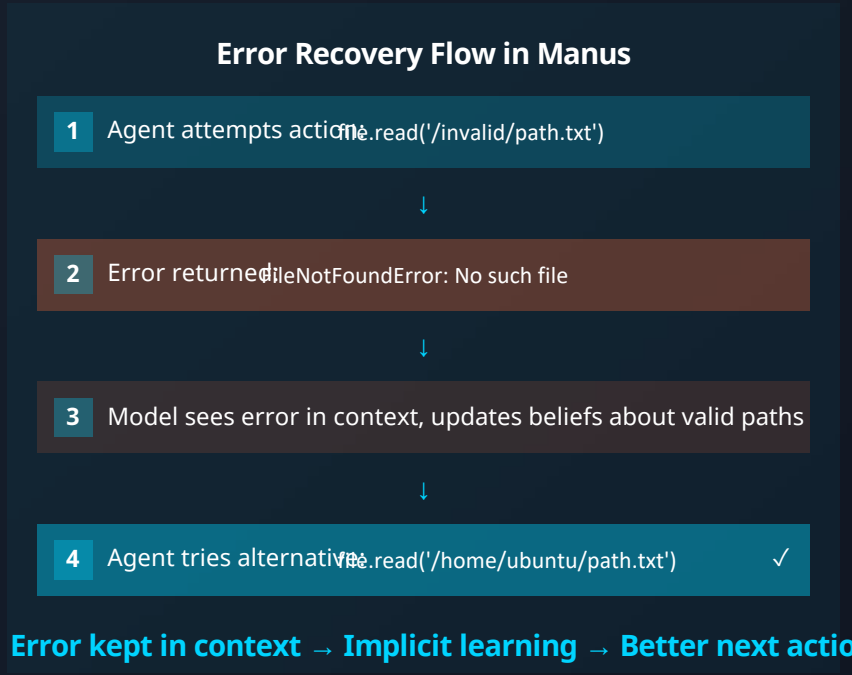
Keep the Errors: They're Evidence

One of the most effective ways to improve agent behavior is **counterintuitive**: leave failed actions in context instead of removing them.
When the model sees a failed action plus its stack trace, it **implicitly updates its beliefs** and shifts its prior away from similar actions.

Key Insight

Errors aren't failures to be swept under the rug—they're **training signals** that help the agent learn what doesn't work, making it less likely to repeat the same mistake.

The agent becomes **self-correcting** through exposure to its own failures, building a more robust understanding of the task space.



Lesson 6

Don't Get Few-Shotted. Randomness is a helper

LLMs are trained to predict the next token based on patterns in context. This makes them **powerful pattern matchers** —but also vulnerable to pattern mimicry.

If the context is full of similar patterns, the model will follow that pattern even when it's **no longer optimal** for the current situation.

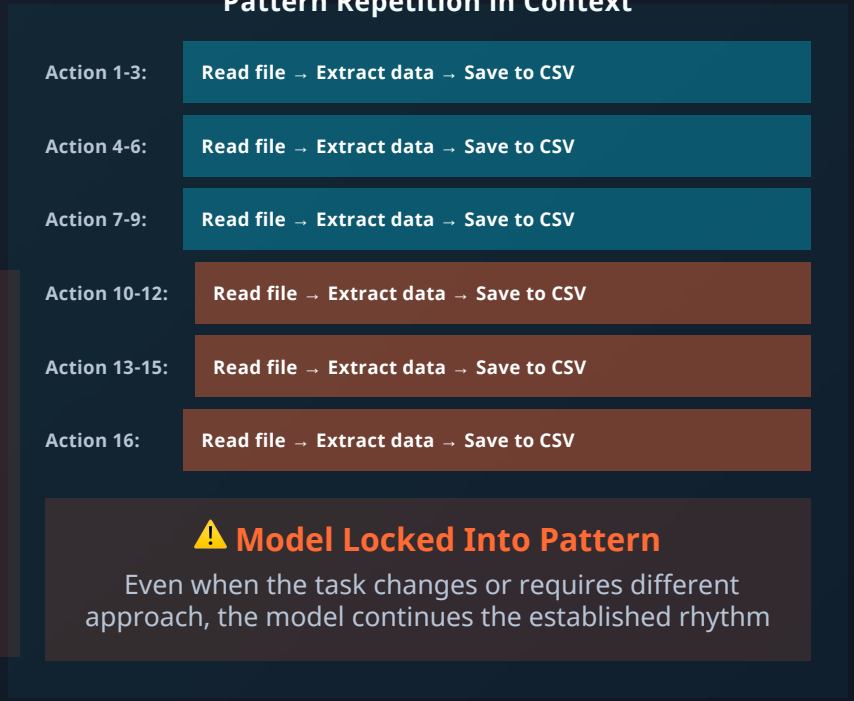
Real-World Example

Task: Review 20 resumes and extract key qualifications.

Resume 1: Open → Read → Extract → Save
Resume 2: Open → Read → Extract → Save
Resume 3: Open → Read → Extract → Save ...
Resume 15: Open → Read → Extract → Save

By resume 15, the agent has fallen into a **mechanical rhythm.** Even if resume 16 requires different handling, the model continues the pattern.

Pattern Repetition in Context



⚠ Model Locked Into Pattern

Even when the task changes or requires different approach, the model continues the established rhythm

Breaking Pattern Mimicry with Structured Variation

Controlled randomness prevents agents from falling into repetitive behavioral loops

Technique 1

Serialization Variation

Use different JSON templates for similar operations to avoid identical formatting

Technique 2

Alternate Phrasing

Vary language in observations and action descriptions to break linguistic patterns

Technique 3

Formatting Noise

Introduce minor variations in whitespace, ordering, and structure

✕ Uniform Pattern

Repetitive Actions

Action 1: {"type": "search", "query": "..."}

Action 2: {"type": "search", "query": "..."}

Action 3: {"type": "search", "query": "..."}

Action 4: {"type": "search", "query": "..."}

Agent stuck in repetitive loop

✓ Varied Pattern

Adaptive Behavior

Action 1: {"type": "search", "q": "..."}

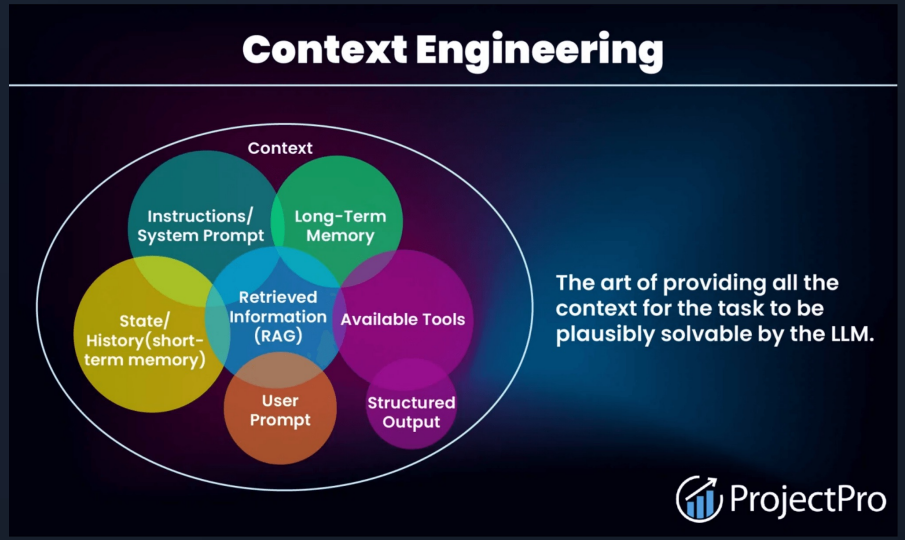
Action 2: {type: "search", query: "..."}

Action 3: {"query": "...", "type": "search"}

Action 4: {"type":"search","query":"..."}

Agent remains adaptable and context-aware

Key Take-away: Context is Everything



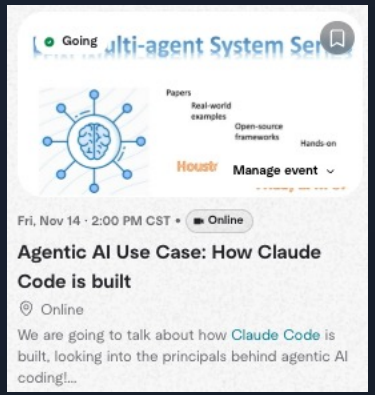
Context engineering is **essential** for building production-ready AI agent systems. How you shape context fundamentally defines how your agent behaves.

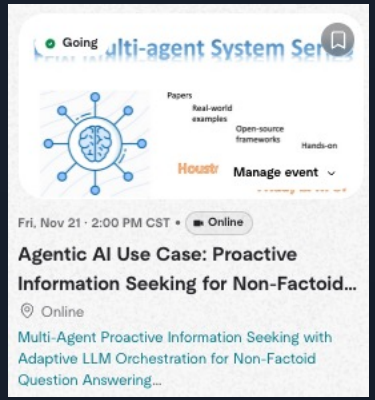
What Context Engineering Controls

- ⚡ **Speed** — KV-cache optimization determines latency and cost
- 🎯 **Accuracy** — Tool masking and attention manipulation guide decisions
- 🔄 **Error Recovery** — Failed actions as evidence improve resilience
- 📈 **Scale** — File system as context enables unlimited growth

Upcoming meetups – Open to proposals and guest speakers!

Nov:





Dec:

Hands-on: Build your first agent!

Slides posted at: https://github.com/YanXuHappygela/LLM-reading-group

Recordings posted at: