Prompt Cache is Architecture.

Not Just an Optimization.

The Comforting Illusion

Like a CDN. Helpful when it hits. Nice for saving a few pennies and reducing latency. Works fine for short, isolated tasks.

The Shocking Reality

Like a structural load-bearing wall. The codebase reveals that cache identity shapes the entirely of the runtime.

The Paradigm Shift

Serious agent systems treat prompt cache as a **structural resource**. If you lose it in long sessions or fan-outs, you change what workflows are economically viable at all.

The Pastel Systems Diary

The Byte-Identical Forking Trick

How runtimes survive multi-agent fan-out.

The Shared Asset

System prompts, schemas, and histories are massive. Rebuilding them for every worker multiplies costs instantly.

Step 1:

Keep the full parent assistant message.

Step 2:

Construct tool_result placeholders.

Step 3:

Give every placeholder the exact same text.

Step 4:

Append the child-specific directive only at the very end.

Why? Byte-Identical Matching.

If workers rebuild the prefix in slightly different shapes, you pay the massive cost three times.

Exact sharing unlocks the ability to afford background tasks and continuous memory extraction.

Approximate is not enough.

The Tradeoff: Extreme Fragility

Cache discipline makes minor refactors dangerous.

The Butterfly Effect

Because the runtime relies on byte-level identity, "equivalent" isn't good enough anymore. Small mismatches matter.

Subtle Regressions

You can quietly destroy your unit economics by tweaking seemingly harmless things:

- Tool ordering
- Prompt rendering
- Model settings
- Summary formatting

The Lesson

You gain massive leverage, but your orchestration design, context design, and cache design are now the same system. Worker topology is tightly coupled to cache strategy.

The Pastel Systems Diary

Redefining Agent Cost

It's not just about the final answer.

The Old View: Token Cost

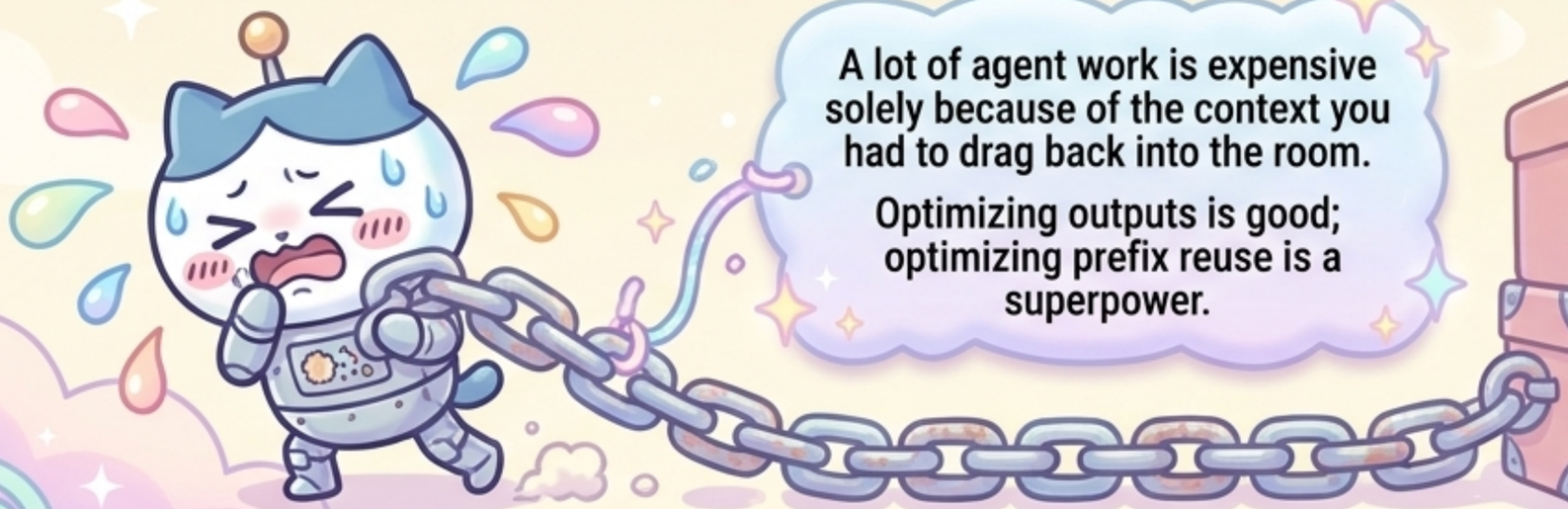
"How many tokens did the model consume to give me the answer?"

➪ An incomplete metric.

The New View: Context Drag

"How much expensive prefix am I forcing the system to rebuild to get there?"

➪ The true design problem.



A lot of agent work is expensive solely because of the context you had to drag back into the room.

Optimizing outputs is good; optimizing prefix reuse is a superpower.

The Pastel Systems Diary

The Architecture Mindset

Building for the future of agent runtimes.

Treat Prefixes as Assets

Expensive context prefixes are reusable inventory, not disposable requests

Design for the Suffix

Fan-out workers so only the absolute smallest end-suffix varies

Guard Your Cache

Assume minor mutations will break cache. Build strict guardrails against formatting changes.

Cost is Runtime Design

Make the economic structure visible to the architecture layer, not just buried in a finance dashboard.

The best agent systems are evolving away from "prompt engineering products" and becoming cache-preserving execution engines with a model inside.