

Start day 06 from day 05. Ignore context of discussion that was transferred to day 08

Thought for 20s >

## 30 day AI experiment – day 06

### Context Carried Over from Day 05

From day 05:

- Goals: AI model evaluation and user interaction pattern refinement
- Technical state: initial tuning, input/output handling tests
- Decisions: feature prioritization and metric selection
- Unresolved:
  - Response consistency
  - Evaluation criteria clarity
  - Systematic measurement of improvement

Day 08 branch is intentionally excluded.

---

## Day 06 – Stabilization and Measurability

Day 06 should shift from experimentation to controlled iteration.

### 1. Objective for Day 06

Move from:

"Testing behavior"

To:

"Measuring and constraining behavior"

Primary focus:

- Reduce variance in responses
  - Formalize evaluation rubric
  - Establish reproducible test cases
- 

### 2. Response Consistency Strategy

#### A. Define Controlled Test Prompts

Create 10–20 canonical prompts across:

- Technical explanation
- Architecture design
- Debugging
- Opinionated reasoning
- Structured output (markdown, JSON, etc.)

For each prompt:

- Store expected structural characteristics
- Define acceptable variance bounds

This creates a baseline regression set.

---

## B. Structural Constraints

Instead of evaluating semantic quality first, constrain structure:

- Heading hierarchy
- Bullet formatting
- Use of examples
- Presence/absence of citations
- Deterministic sections (e.g., always include "Assumptions")

Consistency improves when structure becomes invariant.

---

## 3. Clarifying Evaluation Criteria

Day 05 identified ambiguity in evaluation.

On day 06, define a scoring model:

### Suggested Evaluation Dimensions

1. Structural Compliance (0–5)
2. Logical Coherence (0–5)
3. Factual Accuracy (0–5)
4. Actionability (0–5)
5. Drift from Constraints (0–5)

Document scoring rules explicitly.

Example:

- "5 in structural compliance" = All required sections present, correct ordering, no extraneous segments.
- "0" = Major violation of format constraints.

This eliminates subjective drift.

---

## 4. Tuning Strategy Refinement

Instead of adjusting everything at once:

### Step 1

Lock prompt format.

### Step 2

Iteratively modify only:

- Instruction granularity
- Constraint specificity
- Examples provided

Measure delta per change.

Do not:

- Change multiple variables per iteration.
  - Rely on subjective impressions.
- 

## 5. Deliverables for Day 06

By end of day:

- A fixed regression prompt set
  - A written evaluation rubric
  - At least one measured before/after comparison
  - A log of structural violations observed
- 

## 6. Risk to Monitor

Primary failure mode:

You optimize structure but degrade reasoning depth.

Mitigation:

Introduce one adversarial prompt specifically designed to test reasoning robustness.

---

If you want, we can:

- Draft the regression prompt set
- Formalize the scoring rubric in machine-readable format
- Or implement an automated evaluation harness for it

    ... 

**prompt.yaml**  
File

Use this prompt to write an article discussing my experience. Emphasize that the full no-ai approach of day 06 was intentional and meant to compare while in the middle of high-use

## AI Day 06 – A day of manual work

*The Pareto Line — a 30-day experiment in AI use*

---

### Disclaimer

Each “day” in this series represents a work-day of effort. The posting delay reflects real-world obligations, not retroactive rewriting or assisted optimization. Day 06 was intentionally conducted without AI assistance.

---

### TL;DR

Day 06 was a deliberate no-AI day in the middle of a high-AI-usage experiment.

The goal was comparison, not regression.

Manual work felt slower but more linear.

AI-assisted workflows feel faster but noisier.

The verdict: removing AI mid-stream exposes both the genuine leverage it provides and the structural inefficiencies it introduces. The comparison only makes sense when done inside active usage—not after the fact.

---

### What I Set Out to Do

The primary objective was straightforward: continue building the sandbox manually.

Secondary objectives were more important:

- Compare manual work against AI-assisted work while still immersed in heavy AI usage.
- Observe whether AI usage had already become habitual.
- Prevent masking the ratio of human vs AI effort by reverting entirely to automation.

The build-from-scratch constraint was intentional. If I allowed AI to “help” in small invisible ways, the experiment would blur. Day 06 had to be clean.

---

### What Actually Happened

The work became more linear.

Without AI suggesting branching paths, I approached debugging through experience:

Start at the end.

Verify connectivity.

Measure response time.

Increase verbosity.

Reduce the request to something trivial — “say hi to me.”

Connectivity worked.

cURL returned fast, predictable responses.

The slowdown was not in raw communication.

The friction appeared earlier in the stack: OpenHands introduced excessive context into requests — including large instruction payloads (such as RIS fragments) that were not relevant to the immediate task. The latency was not computational; it was structural. This was not an intelligence problem. It was request bloat.

The paradox remains: building a sandbox so AI can be safely used, using AI, is unnecessarily painful. The abstraction layers accumulate faster than they are justified.

---

## The Hidden Cost of Helpfulness

AI tools increasingly generate their own documentation using AI. The tone is confident.

The structure is impressive. The human optimization is weak.

Automatically generated scaffolding looks complete, but it is often not optimized for how engineers reason under time pressure. It assumes patience and unlimited cognitive bandwidth.

Chatbot helpers are useful.

Their generated structures are frequently sub-par.

This becomes visible only when you remove them and compare.

Manual work exposes friction that AI hides by absorbing it through verbosity.

---

## Where Things Broke

The most significant issue was unsolicited context.

OpenHands injected material into requests that expanded payload size without adding task relevance. That is a textbook example of AI bloat: assistance that increases cost without increasing precision.

The first search for a lightweight alternative led to **OpenClaw** (<https://github.com/razorx89/openclaw>). It was not the right fit — more oriented toward chat interactions than focused sandbox execution.

Its documentation suffered from the same issue identified earlier: optimized for completeness, not for constrained human cognition.

Work then began toward integrating **OpenCode** (<https://github.com/opencode-ai/opencode>) instead.

By the end of the day:

- A simple Ubuntu container was introduced.
- The container can be rebuilt deterministically.
- No local AI agent is fully integrated yet.

This is not accidental. It is transitional.

---

## Rebalancing Effort

Manual debugging was slower in wall-clock time but more predictable.

AI-assisted debugging feels faster but involves lateral exploration. It suggests options. It expands context. It consumes tokens. It generates scaffolding that may not be structurally optimal.

The key difference:

Human work is linear.

AI-assisted work is branching.

Experience enables targeted interventions without random exploration. AI sometimes simulates exploration until something works.

Neither approach is inherently superior. The cost model differs.

---

## Evaluation

**Structural clarity:** Higher without AI.

**Iteration speed:** Higher with AI.

**Noise ratio:** Lower without AI.

**Cognitive overhead:** Mixed — AI reduces typing but increases validation burden.

The most relevant insight:

Blind trust in tools remains a mistake.

AI agents, especially paid ones, would have multiplied costs here. Without visibility into request composition, unnecessary payload inflation would have directly translated to expense.

Day 06 exposed that risk before scaling it.

---

## Results Summary

- No local AI agent created yet.
- Communication pipeline validated.
- Excessive context injection identified as performance bottleneck.
- Cleaner container baseline established.
- Alternative tooling exploration initiated.

This is not failure. It is calibration.

The experience resembles earlier professional situations where friction analysis produced durable knowledge later applied under pressure.

The process is still enjoyable. The learning curve is real. The absence of immediate success does not invalidate the trajectory.

---

## Cost Estimates (Illustrative)

Assume:

- 5 exploratory AI iterations per debugging cycle.

- 15k tokens per expanded request (including unsolicited context).
- \$10 per 1M tokens (representative paid model rate).

Cost per cycle:

$$15,000 \times 5 = 75,000 \text{ tokens}$$

$$75,000 / 1,000,000 \times \$10 = \$0.75 \text{ per debugging cycle}$$

If 40 such cycles occur:

$$\$0.75 \times 40 = \$30$$

This excludes:

- Retry amplification
- Context growth over time
- Agent orchestration overhead

The cost is not catastrophic.

But inefficiency compounds.

Visibility matters before scale.

---

## Links

- [The Pareto Line — 30-day AI experiment \(project repository\)](#) ↗
  - [Articles archive \(prompts, drafts, and published versions\)](#) ↗
  - OpenClaw — <https://github.com/razorx89/openclaw> ↗
  - OpenCode — <https://github.com/opencode-ai/opencode> ↗
- 

## Continuity Hook

Tomorrow (Day 07), I will write a retrospective separating:

- Time genuinely saved because of AI
- Time lost because of AI

Not impressions.

Measured differences.

---

## Disclosure

This experiment tracks real working days.

Day 06 was intentionally AI-free.

The goal was comparison under live conditions — not nostalgia for manual workflows.

Locking this version as published.



Give me the article in downloadable markdown

Download your markdown file here:

[Download AI Day 06 \(Markdown\)](#) «»

day-06.md

File

\* <https://csolomonides.substack.com/p/ai-day-06-a-day-of-manual-work>  
\* <https://anthropocentricsoftware.wordpress.com/2026/02/12/ai-day-06-a-day-of-manual-work/>  
Use the prompt and the links to generate a linkedin post.  
Apply the appropriate RIS

Day 06 — LinkedIn Post (plaintext, light emoji use, punchy hook)

---

I turned AI off on purpose.

Right in the middle of heavy usage.

Day 06 of my 30-day AI experiment was intentionally manual — no assistants, no agent loops, no “just let it try again.”

Why?

Because comparison only makes sense when you’re still inside the system.

If you go back to manual work after weeks of AI, nostalgia contaminates the result.

If you remove AI mid-stream, you see what it actually changed.

What changed?

Manual work was slower.

But it was linear.

AI-assisted work feels faster.

But it branches, expands context, and often injects noise.

The biggest discovery wasn’t speed.

It was payload bloat.

A simple “say hi to me” request became a massive instruction bundle once routed through the tooling stack. The local model wasn’t slow — it was busy parsing irrelevant constraints.

That’s AI bloat:

Assistance that increases cost without increasing precision.

I also noticed something else:

AI-generated documentation is structurally impressive but often optimized for machines, not humans. It assumes infinite patience. It assumes chatbots will fill in the gaps.

Traditional documentation had “rubber duck” friction.

AI documentation removes that friction — and sometimes removes clarity with it.

Day 06 did not produce a working sandbox.

It did produce:

- A validated communication pipeline
- A deterministic container baseline
- Removal of OpenHands + LiteLLM approach
- A clearer understanding of where token costs quietly multiply

Manual debugging is predictable.

AI debugging is iterative.

Humans are the bottleneck.

Machines don't need sleep — but they do need guardrails.

Full write-up here:

Substack:

<https://csolomonides.substack.com/p/ai-day-06-a-day-of-manual-work> ↗

WordPress:

<https://anthropocentricsoftware.wordpress.com/2026/02/12/ai-day-06-a-day-of-manual-work/> ↗

Tomorrow (Day 07):

Measured retrospective — time saved vs time lost.

No impressions. Numbers.

#AI #Engineering #Automation #SoftwareDevelopment #TheParetoLine



Reduce machine smell. Bring it more in line with the article to modes of speech.

I turned AI off on purpose.

Right in the middle of using it heavily.

Day 06 of my 30-day experiment was done fully manually. No assistants. No agents. No "let it try again." That wasn't frustration. It was deliberate.

If you want to compare properly, you don't step away from AI for a week and then romanticize manual work. You remove it while you're still relying on it.

What changed?

The work felt slower.

But it was calmer.

Without AI suggesting five alternative paths at once, I debugged the way I've done for years:

Start from the end of the chain.  
Reduce the request to something trivial.  
Increase verbosity.  
Measure response time.  
Move one layer up.

cURL was fast.  
Ollama was fast.  
LiteLLM was fast.

The slowdown came from injected context higher up the stack. A simple “say hi to me” became a bloated request full of constraints and payload I didn’t ask for. The model wasn’t slow — it was working through noise.

That was the real insight of the day.

AI doesn’t just help. It can quietly expand scope. Expand tokens. Expand cost. Sometimes without increasing precision.

I also noticed something uncomfortable.

A lot of modern tooling documentation is AI-generated. It looks polished. It sounds confident. But it often lacks the “rubber duck” effect human-written docs used to have — where explaining something forced the author to confront their own gaps.

The result is documentation that’s structurally impressive but not always cognitively optimized.

Day 06 didn’t give me a working sandbox.

It gave me something more useful:

- A cleaner container baseline
- A validated communication pipeline
- Removal of an approach that was adding unnecessary complexity
- A clearer view of where cost and friction actually live

Manual debugging is linear.

AI-assisted debugging is iterative.

Humans are slower but predictable.

Machines are tireless but need tight constraints.

Neither is superior. The trade-offs are real.

Full write-up here:

Substack:

[https://csolomonides.substack.com/p/ai-day-06-a-day-of-manual-work ↗](https://csolomonides.substack.com/p/ai-day-06-a-day-of-manual-work)

WordPress:

[https://anthropocentricsoftware.wordpress.com/2026/02/12/ai-day-06-a-day-of-manual-work/ ↗](https://anthropocentricsoftware.wordpress.com/2026/02/12/ai-day-06-a-day-of-manual-work/)

Day 07 will be a measured retrospective: where AI genuinely saved time — and where it quietly cost it.

Use less grandiose tone.

On Day 06, I turned AI off.

Not because it failed. Not because I was frustrated.

But because I wanted a clean comparison.

This experiment is happening while I'm actively using AI every day. So the only way to compare manual vs AI-assisted work fairly is to remove AI in the middle of that usage — not after stepping away for a while.

What changed?

The work felt slower.

But it felt simpler.

Without suggestions branching in different directions, I debugged the way I normally would:

Reduce the request to something trivial.

Start at the last node in the chain.

Increase verbosity.

Measure response time.

Move step by step.

cURL was fast.

Ollama was fast.

LiteLLM was fast.

The slowdown came from injected context further up the stack. A simple request accumulated extra instructions and constraints along the way. The model wasn't struggling — it was processing unnecessary material.

That was the main takeaway.

AI doesn't just help. It can also quietly expand scope — and cost — if you don't monitor what gets sent.

I also noticed something else.

A lot of tooling documentation today is AI-generated. It looks structured and complete, but it isn't always optimized for how engineers actually read under time pressure. Older documentation often had a "rubber duck" quality — the author clarified things while explaining them. That effect is harder to see now.

Day 06 did not result in a working sandbox.

It did result in:

- A validated communication pipeline
- A cleaner container baseline
- Removal of an approach that was adding complexity

- A clearer understanding of where friction was introduced

Manual debugging is linear.

AI-assisted debugging is iterative.

Both have trade-offs.

Full article here:

Substack:

<https://csolomonides.substack.com/p/ai-day-06-a-day-of-manual-work> ↗

WordPress:

<https://anthropocentricsoftware.wordpress.com/2026/02/12/ai-day-06-a-day-of-manual-work/> ↗

Day 07 will focus on measurement — where AI actually saved time, and where it didn't.



Add the disclaimer about delay in posting at the end

On Day 06, I turned AI off.

Not because it failed. Not because I was frustrated.

But because I wanted a clean comparison.

This experiment is happening while I'm actively using AI every day. So the only way to compare manual vs AI-assisted work fairly is to remove AI in the middle of that usage — not after stepping away for a while.

What changed?

The work felt slower.

But it felt simpler.

Without suggestions branching in different directions, I debugged the way I normally would:

Reduce the request to something trivial.

Start at the last node in the chain.

Increase verbosity.

Measure response time.

Move step by step.

cURL was fast.

Ollama was fast.

LiteLLM was fast.

The slowdown came from injected context further up the stack. A simple request accumulated extra instructions and constraints along the way. The model wasn't struggling — it was processing unnecessary material.

That was the main takeaway.

AI doesn't just help. It can also quietly expand scope — and cost — if you don't monitor what gets sent.

I also noticed something else.

A lot of tooling documentation today is AI-generated. It looks structured and complete, but it isn't always optimized for how engineers actually read under time pressure. Older documentation often had a "rubber duck" quality — the author clarified things while explaining them. That effect is harder to see now.

Day 06 did not result in a working sandbox.

It did result in:

- A validated communication pipeline
- A cleaner container baseline
- Removal of an approach that was adding complexity
- A clearer understanding of where friction was introduced

Manual debugging is linear.

AI-assisted debugging is iterative.

Both have trade-offs.

Full article here:

Substack:

[https://csolomonides.substack.com/p/ai-day-06-a-day-of-manual-work ↗](https://csolomonides.substack.com/p/ai-day-06-a-day-of-manual-work)

WordPress:

[https://anthropocentricsoftware.wordpress.com/2026/02/12/ai-day-06-a-day-of-manual-work/ ↗](https://anthropocentricsoftware.wordpress.com/2026/02/12/ai-day-06-a-day-of-manual-work/)

Day 07 will focus on measurement — where AI actually saved time, and where it didn't.

---

*As with the rest of the series, each "day" represents a real workday of effort. Posting may be delayed due to normal obligations, but the work itself follows the actual timeline.*



Give me a download link for the text file

Download the text file here:

[Download AI Day 06 – LinkedIn Post \(.txt\)](#) ↵



