50k LOC of AI Slop And What I've Learned From It

Maciej Bajor

Data Engineer, SEB

DATA SCIENCE SUMMIT AI EDITION

A lecture selected by a Program Council consisting of recognized leaders in the IT and Data Science field.

Warsaw,
18.06.2026 - 19.06.2026

OFFICIAL LECTURE OF THE DATA SCIENCE SUMMIT MACHINE LEARNING EDITION

Event organized by: Academic Partners Foundation

Some research claims that AI is making you worse

- METR (2025):
  Experienced open-source developers took 19% longer with AI, despite believing AI had sped them up.
  https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/

- Shen & Tamkin (2026):
  AI use impaired conceptual understanding, code reading, and debugging, with no significant average efficiency gain.
  https://arxiv.org/abs/2601.20245

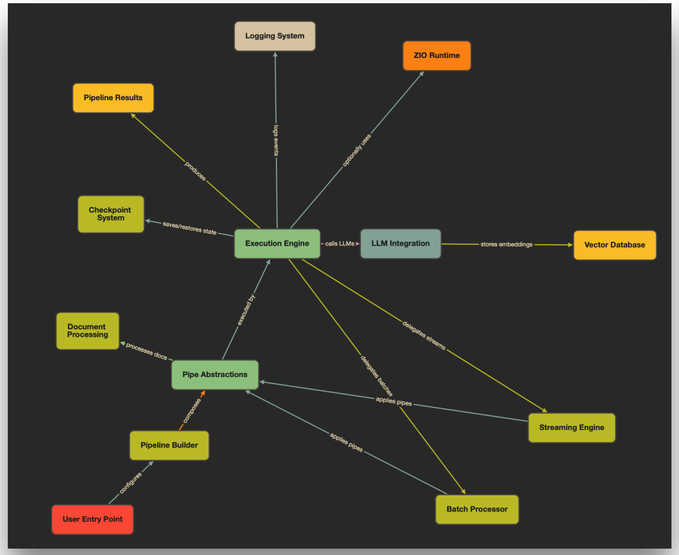
and it's probably true, but on the other hand, it's too powerful tool to ignore

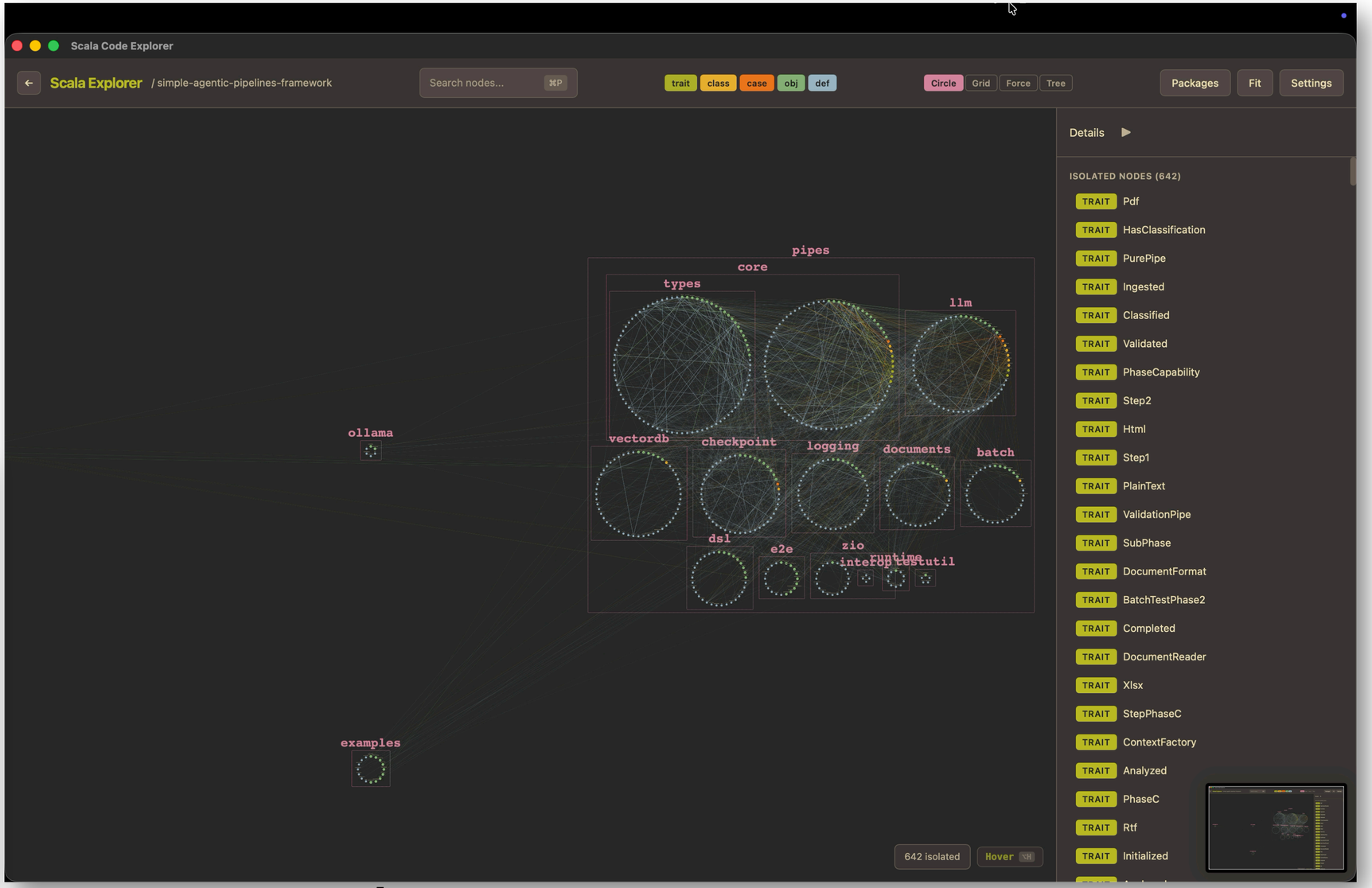
here are my mistakes and habits
I have developed from them
to stop using AI like this:

Code Explorer

It started simple, and I kept adding features by gluing things together with AI.

It mostly worked, but over time I wanted to make it faster and more efficient.





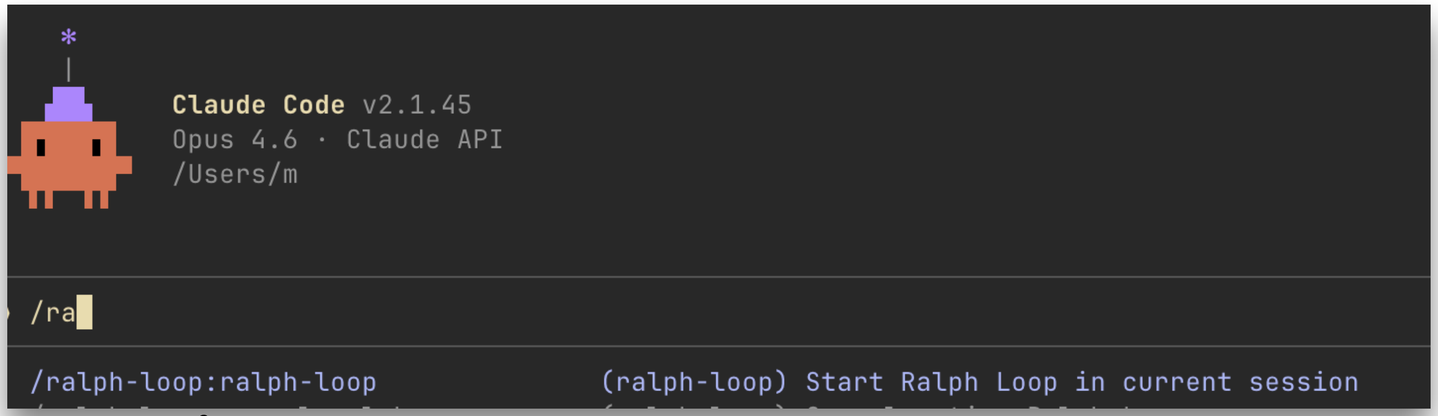
Solution? Obviously, rewrite in Rust.

Not even a joke. That's what I did.

Massive, complex refactor. And my approach?

"Just vibe it bro"

And this is what I ended up with...



git show --stat ead73b7

make tauri-dev ⌘1 | git show --stat ead73b7 ⌘2 | .../repos/archive/scala-code-explorer-mac-os-app ⌘3

```
    Co-Authored-By: Claude Opus 4.5 <noreply@anthropic.com>

 CLAUDE.md                                   |  117 ++++++++----
 Makefile                                    |   22 ++-
 package-lock.json                           |  175 +++--------------
 package.json                                |    3 +-
 src-tauri/Cargo.lock                        |  248 ++++++++++++++++++++++++-
 src-tauri/Cargo.toml                        |   15 ++
 src-tauri/src/algorithms/mod.rs             |    5 +
 src-tauri/src/algorithms/paths.rs           |  175 ++++++++++++++++++
 src-tauri/src/algorithms/reachability.rs    |  167 +++++++++++++++++
 src-tauri/src/algorithms/subgraph.rs        |  442 ++++++++++++++++++++++++++++++++++++++++++++
 src-tauri/src/algorithms/tarjan.rs          |  260 ++++++++++++++++++++++++++
 src-tauri/src/algorithms/topological.rs     |  174 ++++++++++++++++++
 src-tauri/src/analysis/call_edge_builder.rs |  305 +++++++++++++++++++++++++++++++
 src-tauri/src/analysis/entry_points.rs      |  300 ++++++++++++++++++++++++++++++
 src-tauri/src/analysis/instantiation.rs     |  238 ++++++++++++++++++++++++
 src-tauri/src/analysis/mod.rs               |    4 +
 src-tauri/src/analysis/type_hierarchy.rs    |  498 ++++++++++++++++++++++++++++++++++++++++++++++++++
 src-tauri/src/commands.rs                   |  276 ++++++++++++++++++++++++++++
 src-tauri/src/graph/builder.rs              |  244 +++++++++++++++++++++++++
 src-tauri/src/graph/mod.rs                  |    4 +
 src-tauri/src/graph/package_hierarchy.rs    |  239 ++++++++++++++++++++++++
 src-tauri/src/graph/symbol_mapper.rs        |  321 ++++++++++++++++++++++++++++++++
 src-tauri/src/graph/types.rs                |  209 +++++++++++++++++++++
 src-tauri/src/lib.rs                        |   43 ++++-
 src-tauri/src/progress.rs                   |   23 ++
 src-tauri/src/repo/build_parser.rs          |  328 +++++++++++++++++++++++++++++++++
 src-tauri/src/repo/detector.rs              |  300 ++++++++++++++++++++++++++++++
 src-tauri/src/repo/mod.rs                   |    2 +
 src-tauri/src/semanticdb/mod.rs             |    3 +
 src-tauri/src/semanticdb/parser.rs          | 1201 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 src-tauri/src/semanticdb/proto.rs           |  393 +++++++++++++++++++++++++++++++++++++++
 src-tauri/src/semanticdb/types.rs           |  805 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 src-tauri/src/util/intern.rs                |  156 ++++++++++++++++
 src-tauri/src/util/mod.rs                   |    1 +
 src/lib/analysis-pipeline.test.ts           |  294 -----------------------------
 src/lib/analysis-pipeline.ts                |  186 ++++++-------------
 src/lib/build-parser.test.ts                |  315 --------------------------------
 src/lib/build-parser.ts                     |  232 -----------------------
 src/lib/call-edge-builder.test.ts           | 1047 ---------------------------------------------------------------------------------------------------------
 src/lib/call-edge-builder.ts                |  468 -----------------------------------------------
 src/lib/call-graph-integration.test.ts      |  597 ------------------------------------------------------------
 src/lib/call-graph-types.ts                 |  117 ------------
 src/lib/entry-points.test.ts                |  601 ------------------------------------------------------------
 src/lib/entry-points.ts                     |  409 -----------------------------------------
 src/lib/graph-algorithms.test.ts            |  727 -------------------------------------------------------------------------
 src/lib/graph-algorithms.ts                 |  379 ++------------------------------------------
 src/lib/graph-builder.test.ts               | 1599 ----------------------------------------------------------------------------------------------------------------------------------------------------------------
 src/lib/graph-builder.ts                    |  312 -------------------------------
 src/lib/protobuf-reader.test.ts             |  338 ----------------------------------
 src/lib/protobuf-reader.ts                  |  189 -------------------
 src/lib/repo-detector.test.ts               |   47 -----
 src/lib/repo-detector.ts                    |  135 ++------------
 src/lib/semanticdb-parser.test.ts           |  395 ----------------------------------------
 src/lib/semanticdb-parser.ts                | 1016 ------------------------------------------------------------------------------------------------------
:
```

Wanted to build a project to understand AI generated code better. Ended up creating unmaintainable mess.

I spotted errors in it and AI was unable to fix it. And good luck debugging 50k LOC you didn't write in the lang you don't know.

Something had to change.

Do not let the AI jump from vague intent to code

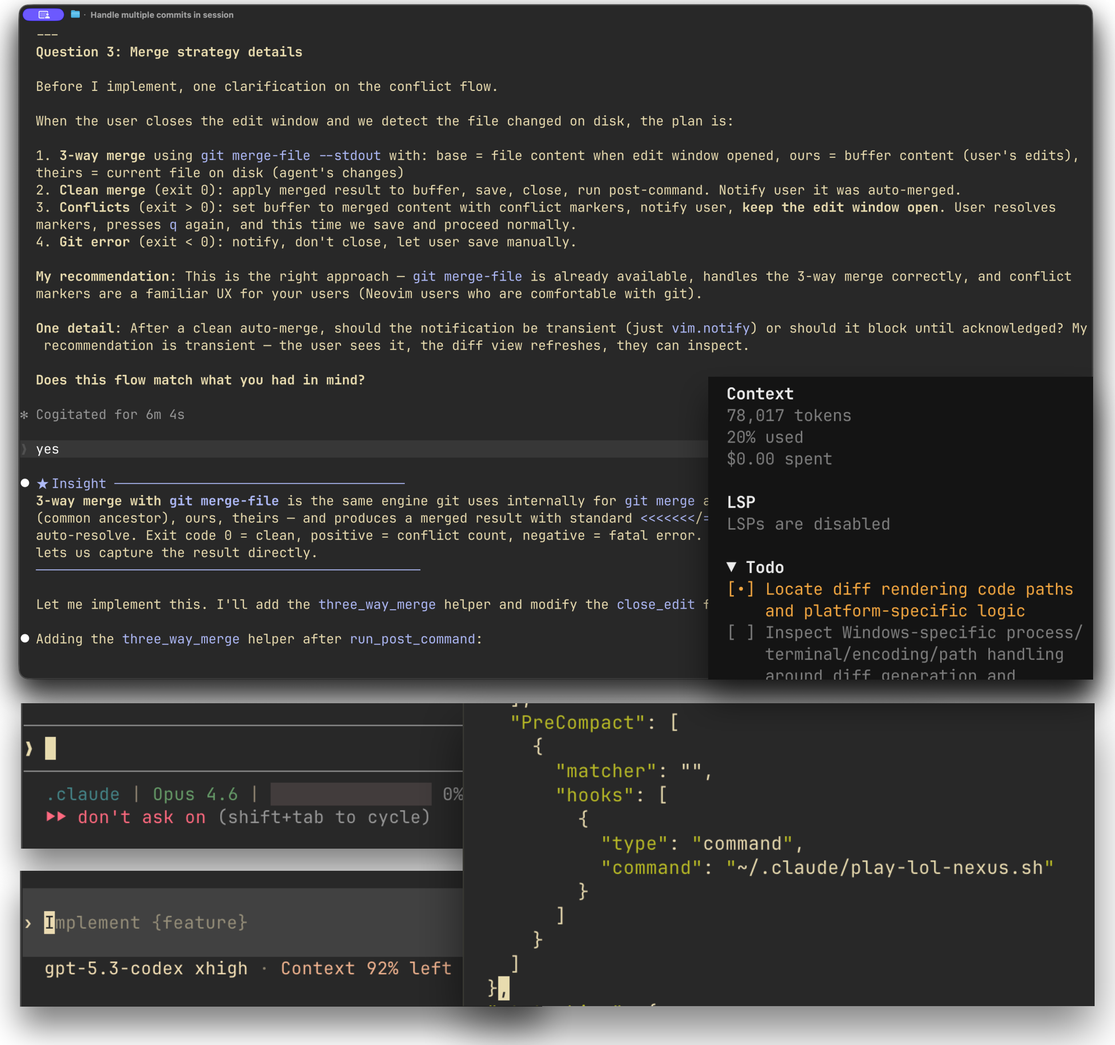
1. **Ask AI to feed itself with relevant context but not create anything yet**
2. **Use plan mode**, consider having superpowers installed: https://github.com/obra/superpowers
3. **If not aligned enough** invoke grill-me skill

https://github.com/mattpocock/skills/blob/main/skills/productivity/grill-me/SKILL.md

4. **Approve or correct the plan**
5. **Only then allow implementation**

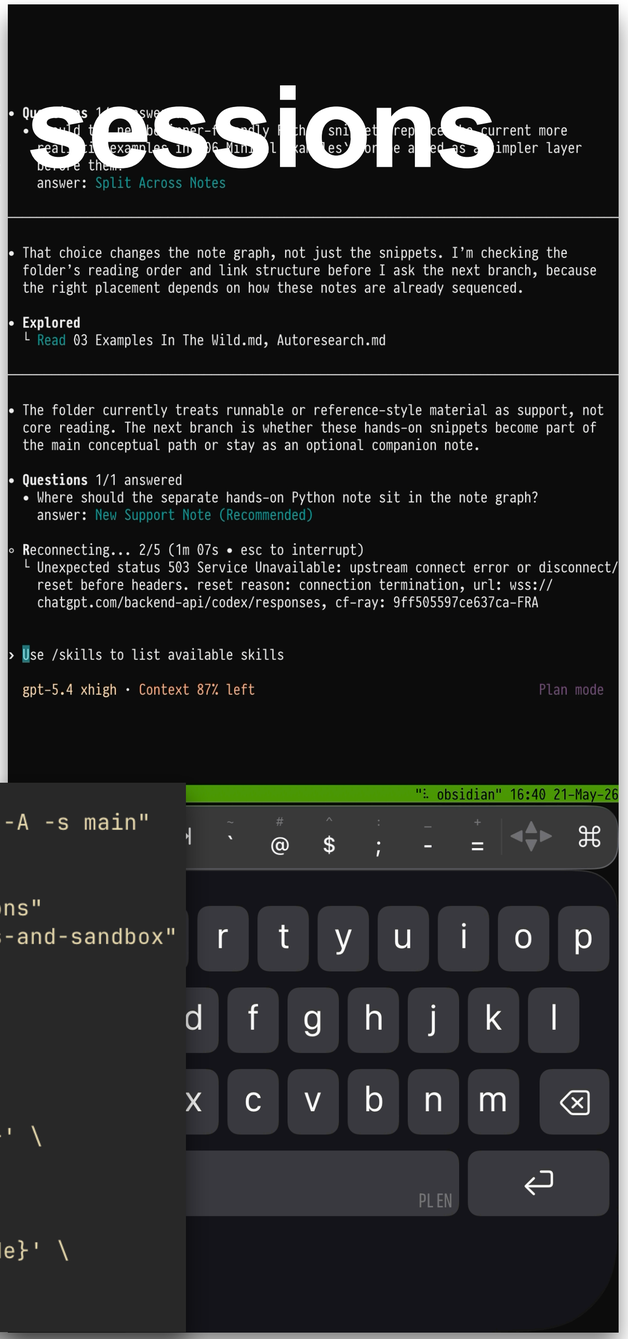
Watch context usage - each compact will compress context.
In Codex you can configure it to see % of context left in /statusline.
In Claude Code you can set a script to play sound triggered by PreCompact hook - in ~/.claude/settings.json.

In OpenCode context usage is visible by default.



Mobile setup for longer alignment sessions

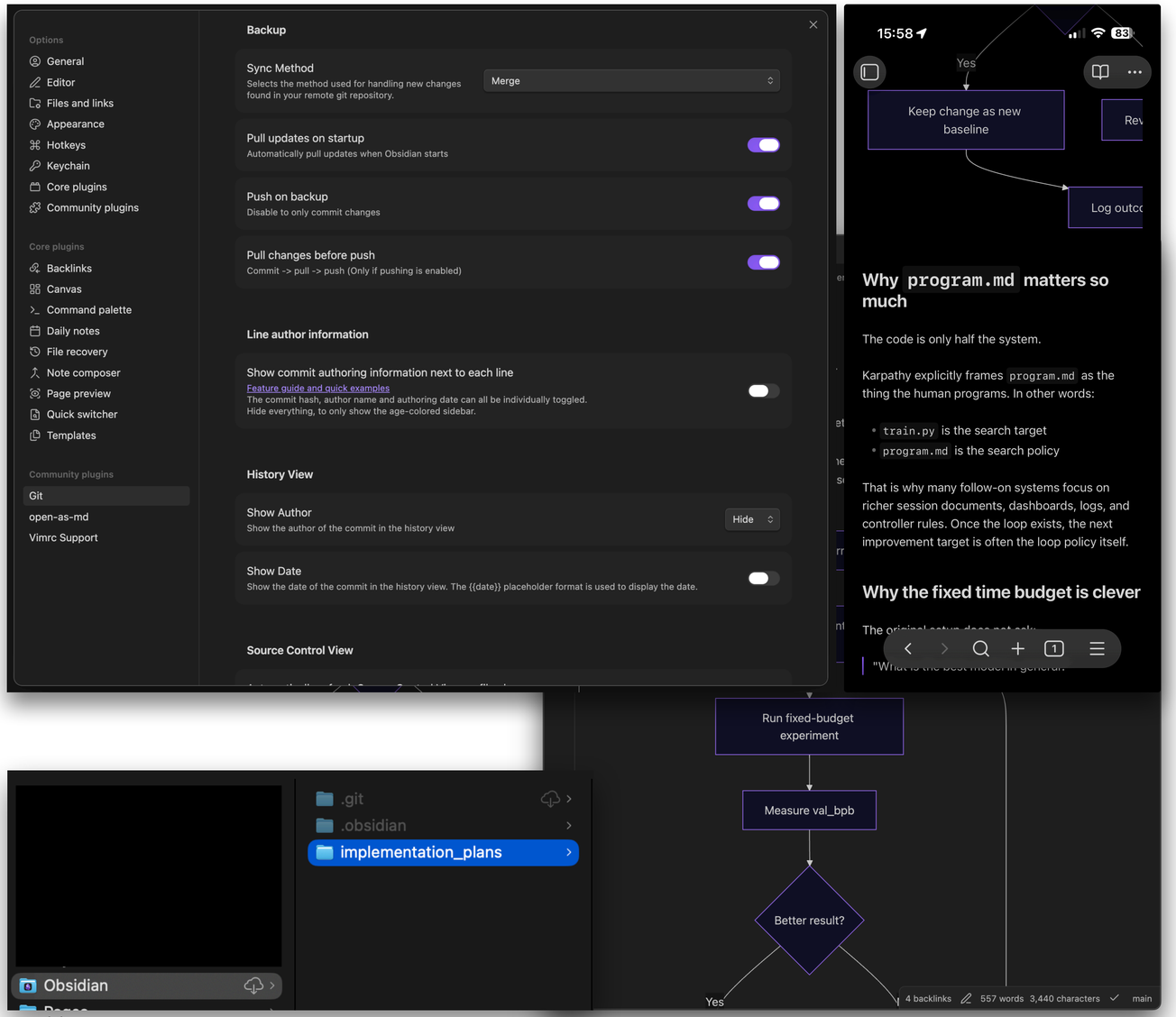
- Phone and computer connected to the same VPS with AI tools installed
- Blink Shell on iOS
- Mosh instead of plain SSH for more stable connection
- Persistent tmux session allows:
  - reconnecting from multiple devices
  - continuing the same terminal session
  - scrolling back through preserved terminal history




```
# on my computer:
alias vps="mosh user@<server-ip> -- tmux new-session -A -s main"

# ~/.bashrc on vps:
alias claudeskip="claude --dangerously-skip-permissions"
alias codexskip="codex --dangerously-bypass-approvals-and-sandbox"

~/.tmux.conf
set -g mouse on
set -g history-limit 100000

# Enable scroll
bind -T root WheelUpPane if-shell -F '#{alternate_on}' \
    'send-keys -M' \
    'copy-mode -e; send-keys -X scroll-up'

bind -T root WheelDownPane if-shell -F '#{pane_in_mode}' \
    'send-keys -X scroll-down' \
    'send-keys -M'
```

Implementation plans with Obsidian

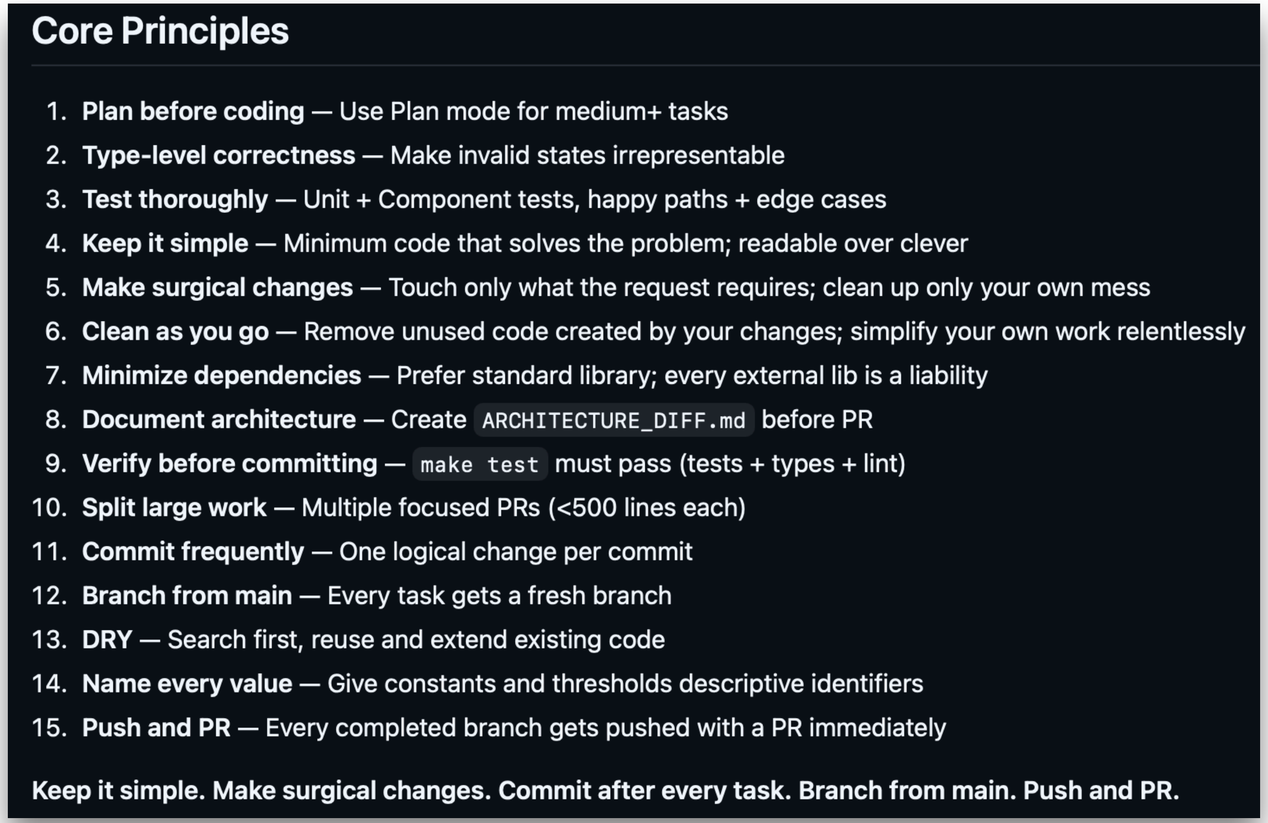
- I use the same VPS setup to **generate notes and implementation plans** for review.
- I ask the agent to format the output for Obsidian.
- Sync flow:
  - **VPS → GitHub → Obsidian on computer → iCloud Drive → phone**
- VPS pushes files to GitHub.
- On the computer, Obsidian Git syncs every N minutes:
  - pulls latest notes
  - commits local changes
  - pushes updates
- The Obsidian vault is stored in iCloud Drive.
- The phone reads the notes through iCloud sync.



AGENTS.md

- Extended existing functionality rather than gluing something new.
- A good sign: it deleted or modified lines instead of just adding new ones.
- Prefer positive structure over negation-only rules.
- Write it around behaviour you want, not mistakes you want to avoid.

Rana (2026)
Negative constraints in LLM prompts often backfire: telling a model “do not use X” can make X more likely.
https://arxiv.org/abs/2601.08070



Core Principles

1. **Plan before coding** — Use Plan mode for medium+ tasks
2. **Type-level correctness** — Make invalid states irrepresentable
3. **Test thoroughly** — Unit + Component tests, happy paths + edge cases
4. **Keep it simple** — Minimum code that solves the problem; readable over clever
5. **Make surgical changes** — Touch only what the request requires; clean up only your own mess
6. **Clean as you go** — Remove unused code created by your changes; simplify your own work relentlessly
7. **Minimize dependencies** — Prefer standard library; every external lib is a liability
8. **Document architecture** — Create `ARCHITECTURE_DIFF.md` before PR
9. **Verify before committing** — `make test` must pass (tests + types + lint)
10. **Split large work** — Multiple focused PRs (<500 lines each)
11. **Commit frequently** — One logical change per commit
12. **Branch from main** — Every task gets a fresh branch
13. **DRY** — Search first, reuse and extend existing code
14. **Name every value** — Give constants and thresholds descriptive identifiers
15. **Push and PR** — Every completed branch gets pushed with a PR immediately

Keep it simple. Make surgical changes. Commit after every task. Branch from main. Push and PR.

Encoding correctness in types

- Don’t validate bad states late — make them unrepresentable.
- Types can encode domain rules the compiler can enforce before tests run.
- This helps with AI-assisted coding: constraints become structural, not advisory.

Sternfeld, Kucharavy (2025):

By prompting LLMs to leverage features such as sealed traits, smart constructors, and refined return types, we enable the generation of programs that encode correctness directly into their type signatures.
https://arxiv.org/html/2510.11151v1

Make invalid states unrepresentable

`birthday(age: int)` → -5 compiles fine. Bug ships.

`birthday(age: Age)` → -5 rejected at compile time.

Works only for compile-time-known values!
Runtime input still needs validation.

```scala
import scala.compiletime.error

// Before: accepts negative age — no compiler help
def birthday(age: Int): String = s"You are $age"

// After: compile-time rejection of negative literals
opaque type Age = Int
object Age:
  inline def apply(inline n: Int): Age =
    inline if n < 0 then error("Age cannot be negative")
    else n

def happyBirthday(age: Age): String = s"You are $age"

@main def demo(): Unit =
  println(birthday(-5))             // compiles fine — bug
  println(happyBirthday(Age(25)))   // works
  println(happyBirthday(Age(-5)))   // compile error
```

CI: clean-up obvious dead code

- Configure CI to fail on unused or unreachable code
- Many linters/compilers can detect obvious unused or unreachable code
- This is not complete dead-code analysis
- Public APIs and dynamic entry points
  may look unused but still be required

```yaml
name: CI

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  lint:
    name: Luacheck
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Install Luacheck
        run: |
          sudo apt-get update
          sudo apt-get install -y lua5.1 luarocks
          sudo luarocks install luacheck

      - name: Run Luacheck
        run: luacheck lua/ plugin/ --config .luacheckrc
```

```
-- Enable dead code detection
unused = true
unused_args = true
unused_secondaries = true
```

CI: testing the tests

- Mutation testing temporarily changes production code, then reruns the tests.
- If tests still pass, the mutant survived — the tests may be weak.
- **Heavy by design:**
  - tests may run once per mutation
  - best used selectively: risky code, core logic, or changed files
  - surviving mutants need review: weak tests, dead code, unclear requirements,

Wang, Xu, Briand, Liu (2026):

some test suites achieve 100% coverage but only 4% mutation score

https://arxiv.org/abs/2506.02954

plan.json 〉 No Selection

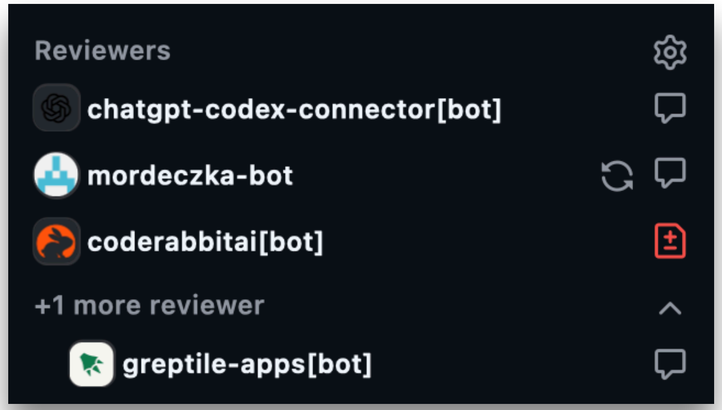

```
},
{
  "column": 40,
  "end_offset": 3026,
  "file_path": "lua/raccoon/config.lua",
  "line": 99,
  "mutant_id": "49495ae3dd2b",
  "operator": "logical_connector",
  "ordinal": 19,
  "original": "or",
  "replacement": "and",
  "selected_specs": [
    "tests/config_compat_spec.lua",
    "tests/config_spec.lua",
    "tests/keymaps_spec.lua"
  ],
  "source_excerpt": "if type(value.token) ~= \"string\" or value.token == \"\" then",
  "start_offset": 3024
},
```


status	mutant	line:col	operator	change
killed	9fb85e390264	91:95	nil_guard_invert	~= -> ==
killed	6dfde847cf15	98:20	comparison_equality	== -> ~=
killed	49495ae3dd2b	99:40	logical_connector	or -> and
killed	f5bf17122852	102:21	nil_guard_invert	~= -> ==
killed	14eb0b00ab8b	103:16	boolean_literal	false -> true

AI "reviewers" as filters, not actual reviewers

Good at:

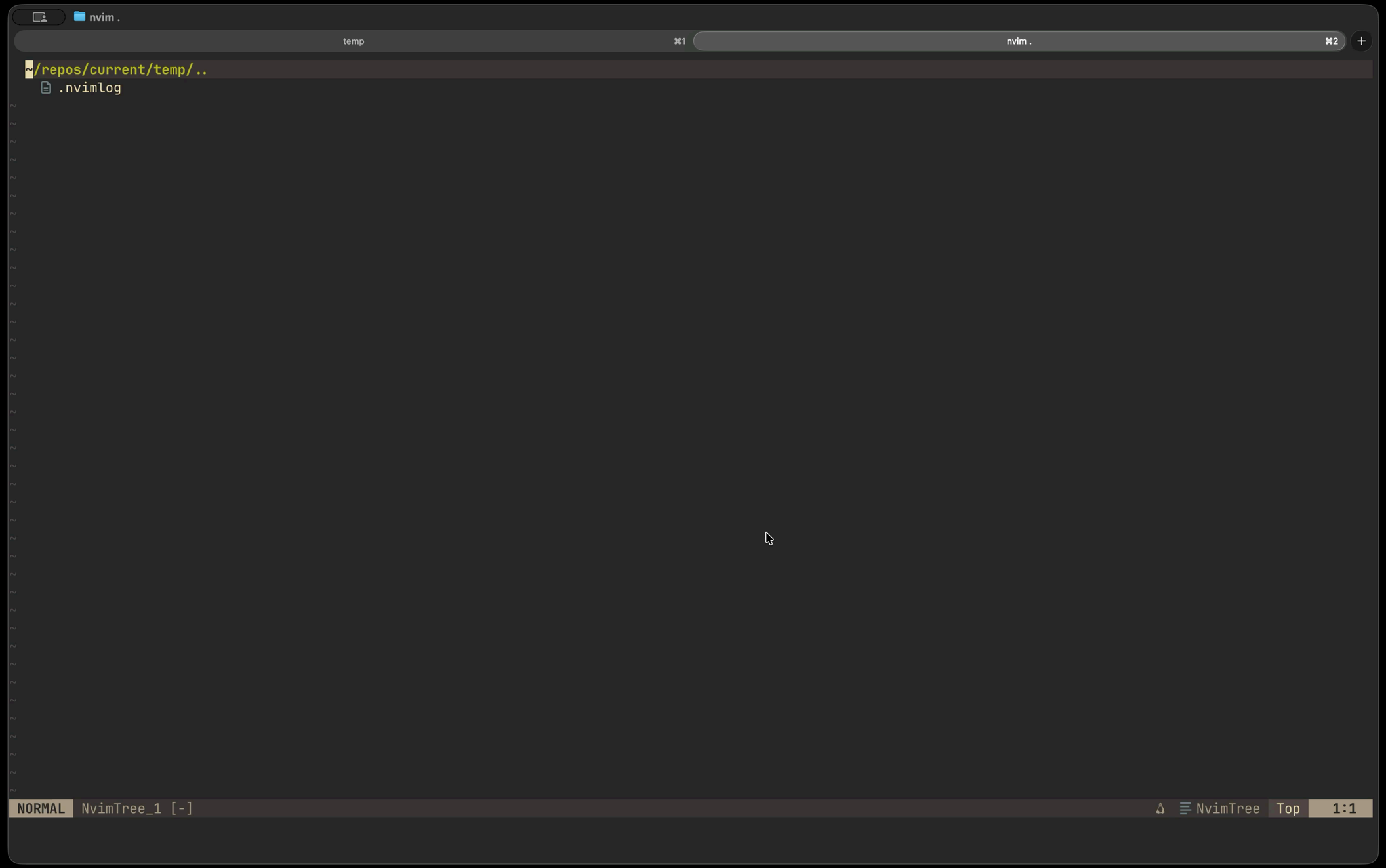
- **“feasible and has a positive impact on the developer workflow”**
  Google AutoCommenter — https://arxiv.org/abs/2405.13565
- **“more adept at detecting functional errors”**
  SWR-Bench — https://arxiv.org/abs/2509.01494
- **“automatic summarization of complex pull requests”**
  Rethinking Code Review Workflows with LLM Assistance — https://arxiv.org/abs/2505.16339
- **“significantly outperform the state-of-the-art static analysis tools”**
  Security Code Review with LLMs — https://arxiv.org/abs/2401.16310



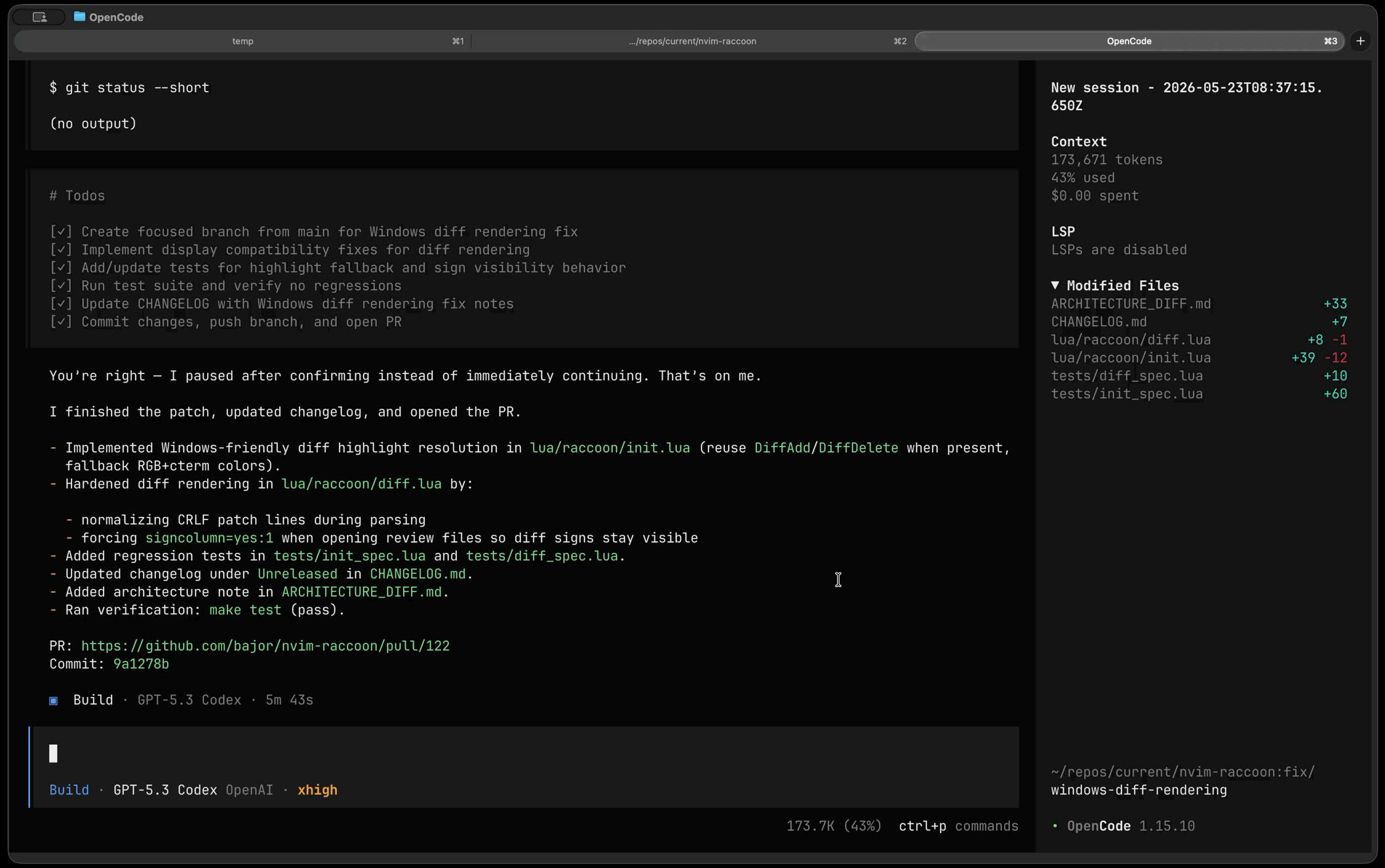
Bad at:

- **“current systems underperform”** *paper from 1 Sept 2025
  SWR-Bench — https://arxiv.org/abs/2509.01494
- **“high false positive rates”**
  SWR-Bench — https://arxiv.org/abs/2509.01494
- **“not identifying more high-severity issues”**
  Deep Learning-based Code Reviews — https://arxiv.org/abs/2411.11401
- **“did not result in saved time”**
  Deep Learning-based Code Reviews — https://arxiv.org/abs/2411.11401
- **“faulty reviews, unnecessary corrections, and irrelevant comments”**
  Automated Code Review in Practice / Qodo PR-Agent — https://arxiv.org/abs/2412.18531
- **“misclassify correct code implementation as non-compliant or defective”**
  Requirement Conformance Study — https://arxiv.org/abs/2603.00539

use whatever tools make reviewing AI-generated code more engaging — or build your own



http://github.com/bajor/nvim-raccoon



https://opencode.ai/

Reading AI generated code with AI generated diagrams

- Asked AI to visually explain what changed in the PR.
- Created a skill that generates Mermaid diagrams explaining code changes.
- Configured AGENTS.md to generate an ARCH_DIFF.md file by default when creating a PR.
- ARCH_DIFF.md includes a short summary and embedded Mermaid diagrams.

nvim-raccoon

nvim-raccoon ⌘1 | nvim SKILL.md ⌘2

```
Tip: Try the Codex App. Run 'codex app' or visit https://chatgpt.com/codex?app-landing-page=true

› $pr-

  pr-diagram                  [Skill] Analyze a GitHub pull request using `gh pr diff`, map changed files and dependency impact, and generate Mermaid diagrams plus a summary. Use when user wants PR analysis, PR …
  Test-Driven Development     [Skill] Implement features and bug fixes with test-first development
  Using Git Worktrees         [Skill] Create isolated git worktrees for parallel feature work
  Using Superpowers           [Skill] Establish how and when to use Superpowers skills
  Superpowers                 [Plugin] An agentic skills framework & software development methodology that works: planning, TDD, debugging, and collaboration workflows.
  Writing Plans               [Skill] Turn specs into detailed multi-step implementation plans
  Writing Skills              [Skill] Create and verify skills before deployment
  Subagent-Driven Development [Skill] Execute implementation plans with staged subagent reviews

  Press enter to insert or esc to close
```

nvim SKILL.md

nvim-raccoon ⌘1 | nvim SKILL.md ⌘2

```
---
name: pr-diagram
description: Analyze a GitHub pull request using `gh pr diff`, map changed files and dependency impact, and generate Mermaid diagrams plus a summary. Use when user wants PR analysis, PR diagrams, blast-radius an
alysis, or mentions `gh pr diff`.
---

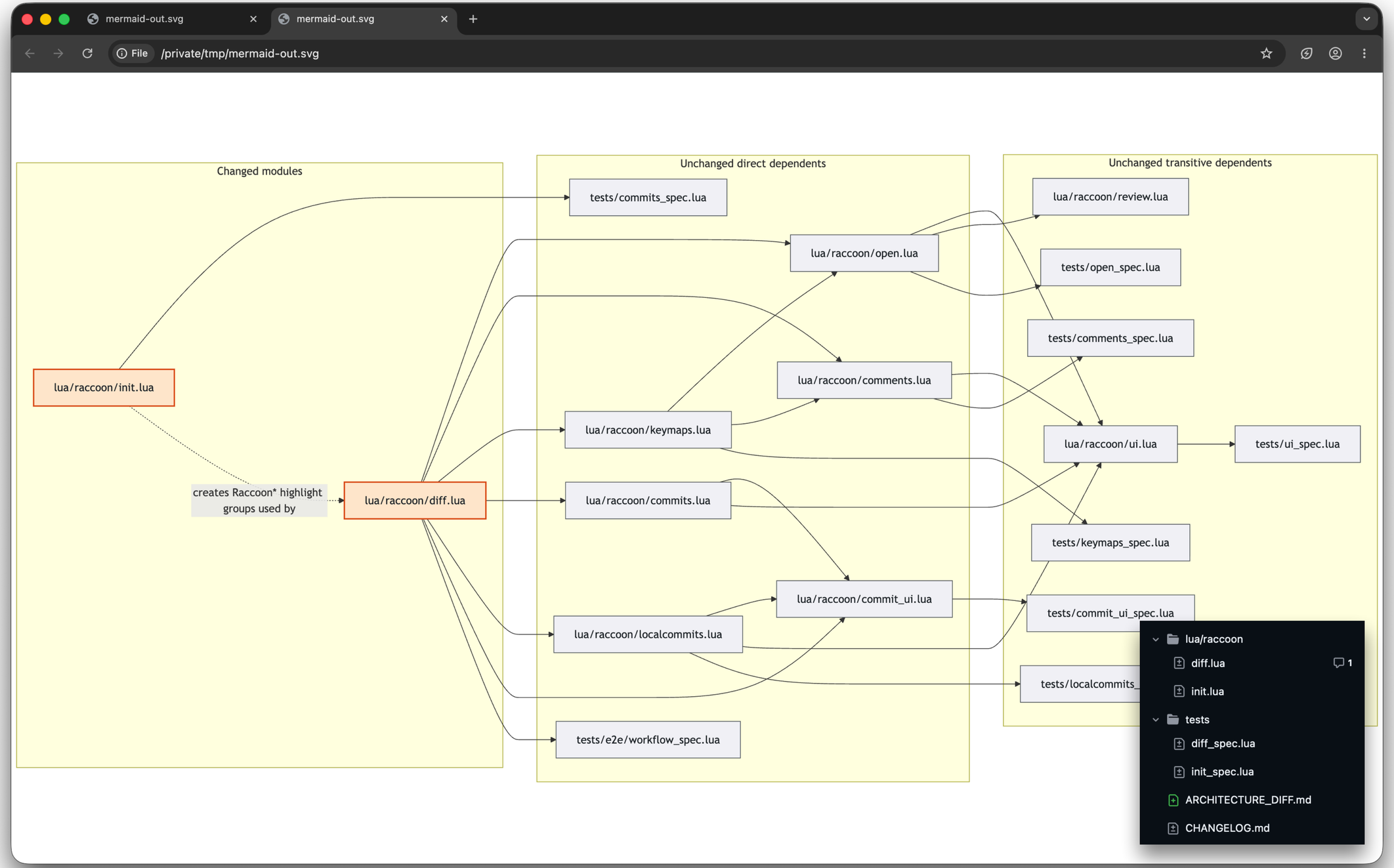
Analyze PR using `gh pr diff`

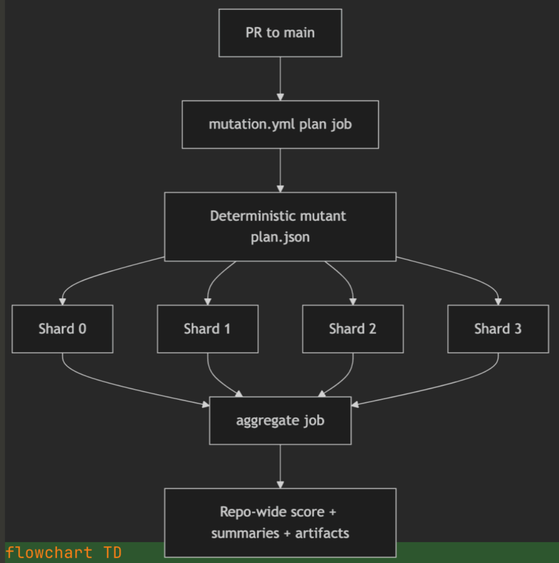
First, create a timestamped output directory using these EXACT commands:
```bash
TIMESTAMP=$(date '+%Y-%m-%d %H-%M-%S')
OUTPUT_DIR="./pr-diagram/${TIMESTAMP}"
mkdir -p "$OUTPUT_DIR"
```
ALL output files MUST be written to `$OUTPUT_DIR` (e.g., `$OUTPUT_DIR/summary.md`).

Identify:
- All changed files and their roles (controller, service, model, config, test, etc.)
- The dependency graph of changed files (what imports what)
- Which unchanged modules depend on the changed ones (blast radius)
- The key behavioral change (before vs after)

Generate THREE Mermaid diagrams, saving each as a separate `.mmd` file in `$OUTPUT_DIR`:
1. **$OUTPUT_DIR/file-changes.mmd** — files changed, grouped by layer/module
2. **$OUTPUT_DIR/blast-radius.mmd** — how changes propagate to unchanged parts of the system. Focus on modules that transitively depend on changed files but are NOT themselves changed.
3. **$OUTPUT_DIR/before-after-flow.mmd** — the main logic change as two sequence diagrams (before and after)

Generate **$OUTPUT_DIR/summary.md** containing:
- PR number and title
- One-paragraph summary of what changed and why it matters
- List of high-risk areas (unchanged code most likely to break)
- Links to the three diagrams (relative links: `./file-changes.mmd`, etc.)

After writing all files, print the full absolute path to `$OUTPUT_DIR` and list its contents with `ls -la "$OUTPUT_DIR"`.
```

```
# Add to ~/.zshrc
mermaid() { mmdc -i "$1" -o /tmp/mermaid-out.svg && open /tmp/mermaid-out.svg; }
```

```
m@MacBook-Air-m 2026-05-24 08-02-59 % ls
before-after-flow.mmd   blast-radius.mmd        file-changes.mmd        summary.md
m@MacBook-Air-m 2026-05-24 08-02-59 % mermaid file-changes.mmd
Generating single mermaid chart
```

NORMAL main SKILL.md utf-8 markdown Top 1:1

https://github.com/bajor/tools-configs/blob/main/skills/pr-diagram/SKILL.md


mermaid-out.svg
mermaid-out.svg
File
/private/tmp/mermaid-out.svg
Changed modules
Unchanged direct dependents
Unchanged transitive dependents
lua/raccoon/init.lua
creates Raccoon* highlight groups used by
lua/raccoon/diff.lua
tests/commits_spec.lua
lua/raccoon/open.lua
lua/raccoon/keymaps.lua
lua/raccoon/comments.lua
lua/raccoon/commits.lua
lua/raccoon/commit_ui.lua
lua/raccoon/localcommits.lua
tests/e2e/workflow_spec.lua
lua/raccoon/review.lua
tests/open_spec.lua
tests/comments_spec.lua
lua/raccoon/ui.lua
tests/ui_spec.lua
tests/keymaps_spec.lua
tests/commit_ui_spec.lua
tests/localcommits_
lua/raccoon
diff.lua
1
init.lua
tests
diff_spec.lua
init_spec.lua
ARCHITECTURE_DIFF.md
CHANGELOG.md

nvim .

nvim . ⌘1 vps ⌘2

```
+ # Architecture Diff
+ ## Summary
+ Add a repo-local mutation testing system with deterministic sampling, explicit module-to-spec mapping, sharded execution, and a dedicated GitHub Actions gate for PRs targeting `main`, starting with a blocking co
  re-module scope and explicit deferred UI-heavy modules in policy.
+
+ ## Diagram(s)
+ ```mermaid
```



AGENTS.md

ines (133 loc) · 6.83 KB

6. Architecture Documentation

Create `ARCHITECTURE_DIFF.md` in repo root before opening a PR. Include at least one Mermaid diagram (flowchart for components, sequence for data flow, ER for schema). Replace any existing one.

```
# Architecture Diff
## Summary
One-sentence description.
## Diagram(s)
```mermaid
graph TD
    A[Existing Service] --> B[New Module] --> C[Database]
```

```
+ flowchart TD
+     A[PR to main] ⟶ B[mutation.yml plan job]
+     B ⟶ C[Deterministic mutant plan.json]
+     C ⟶ D[Shard 0]
+     C ⟶ E[Shard 1]
+     C ⟶ F[Shard 2]
+     C ⟶ G[Shard 3]
+     D ⟶ H[aggregate job]
+     E ⟶ H
+     F ⟶ H
+     G ⟶ H
+     H ⟶ I[Repo-wide score + summaries + artifacts]
+ ```
+
+ ## Changes
+ ### Added
+ - `scripts/mutation/`: Python mutation harness with plan, shard, and aggregate commands.
+ - `mutation/manifest.json`: Explicit module-to-spec mapping with reusable spec groups.
+ - `mutation/policy.json`: CI policy for threshold, timeout, sampling, shard defaults, and temporary excluded modules with rationale.
+ - `mutation/equivalents.json`: Reviewed equivalent-mutant allowlist.
+ - `.github/workflows/mutation.yml`: Dedicated PR-to-`main` mutation workflow with shard artifacts and aggregate summary.
```

NORMAL mutation-testing-ci [3/20] ✓ In sync ARCHITECTURE_DIFF.md [-] utf-8 markdown 16% 6:1

https://github.com/3rd/diagram.nvim
```

lessons learned

Slow down. Seriously.

Small PRs and code review are useful bottlenecks. They make it harder to use AI like a slot machine.

For low-impact work, some misunderstanding may be acceptable. For critical systems people rely on, it is not.

Actively resist the YOLO mindset. If you can’t explain how the critical parts of the system works, reason about the trade-offs, and challenge the AI’s output - you're not moving faster, you’re creating long-term risk.

Good code is small

Easy to forget when generation feels free.

A PR that deletes or modifies lines instead of just adding them is a good sign. Extending existing code rather than gluing new things on the side.

Simple setups last longer

Every custom tool you build is a tool you have to maintain.

Build enough of them and that's all what you're doing.

You do not see all the edge cases and issues until you actually start using it.

There are a lot of tools I built and killed because I found something existing that does the job better. That's a good outcome, not a failure.

But when there is no alternative - build it

There's real value in actually trying things and seeing for yourself what works, what doesn't and why. Even if you delete everything at the end.

The ease of spinning up multiple versions or PoCs fast and choosing between them pushes you toward a higher-level designer or researched role.

Feedback

Zeskanuj kod i zostaw swoją opinię

DATA SCIENCE SUMMIT AI EDITION

50k Lines of AI Slop and What I Learned From It

Maciej Bajor

https://ml.dssconf.pl/user.html#!/lecture/DSSML26-58e6/rate