AI-Assisted Software Engineering

Anthony Panozzo
June 25, 2025

Quick intro

What's interesting about AI-assisted software engineering?

Opportunities:

- Faster value delivery
- Helps with annoying / painful things
  - Maintenance, refactoring, software upgrades / updates
- Wrangle complexity
- Allows us to move up one level of abstraction
- Career: New skill / toolset to learn

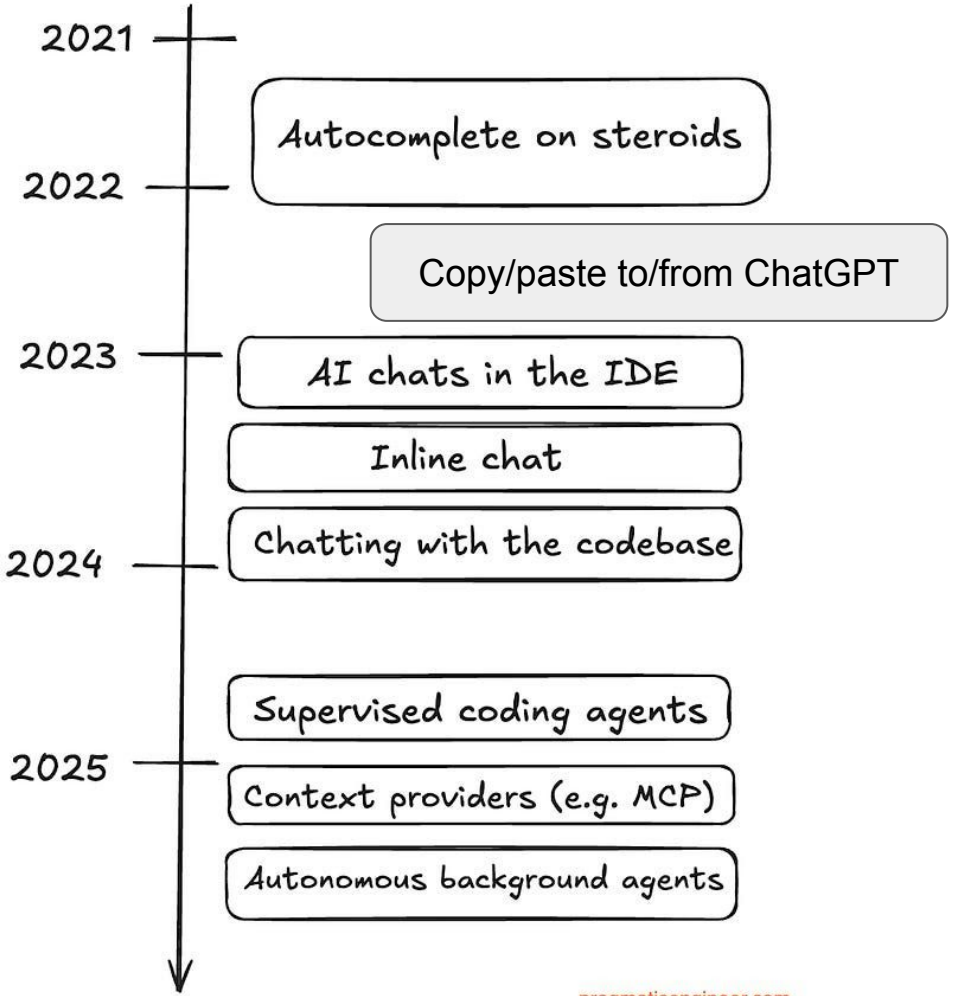
What's interesting about AI-assisted software engineering?

Risks / Questions:

- Will my codebase be maintainable?
- Will we still understand the code?
- Will we introduce security / performance issues?
- Can AI work with my large/complex codebase?
- Is it actually faster?
- It's a fast moving space

"Is there a way to skillfully use AI to deliver value faster while avoiding the downsides?"

Quick overview of tools in this space


2021
Autocomplete on steroids
2022
Copy/paste to/from ChatGPT
2023
AI chats in the IDE
Inline chat
2024
Chatting with the codebase
Supervised coding agents
2025
Context providers (e.g. MCP)
Autonomous background agents

Classic GitHub Copilot

- Fancy autocomplete / tab complete
- Watches recent edits and tries to do something similar
- A feature of most editors now

General purpose chat AI tools

- Examples:
  - OpenAI's ChatGPT
  - Anthropic's Claude
  - Google's Gemini
  - etc.
- Pretty useful for functions, tests, making a plan, etc.
- Downsides:
  - You need to copy/paste your code to give context
  - And to get responses out, copy-pasting again

Embedded AI in IDEs

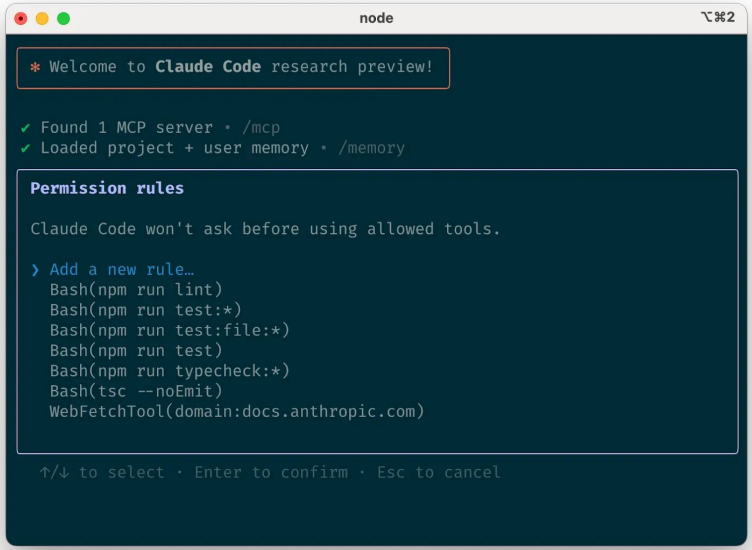
Examples:

- Cursor
- WindSurf
- Cline
- Zed
- VS Code / Copilot

CLI / Standalone AI agents

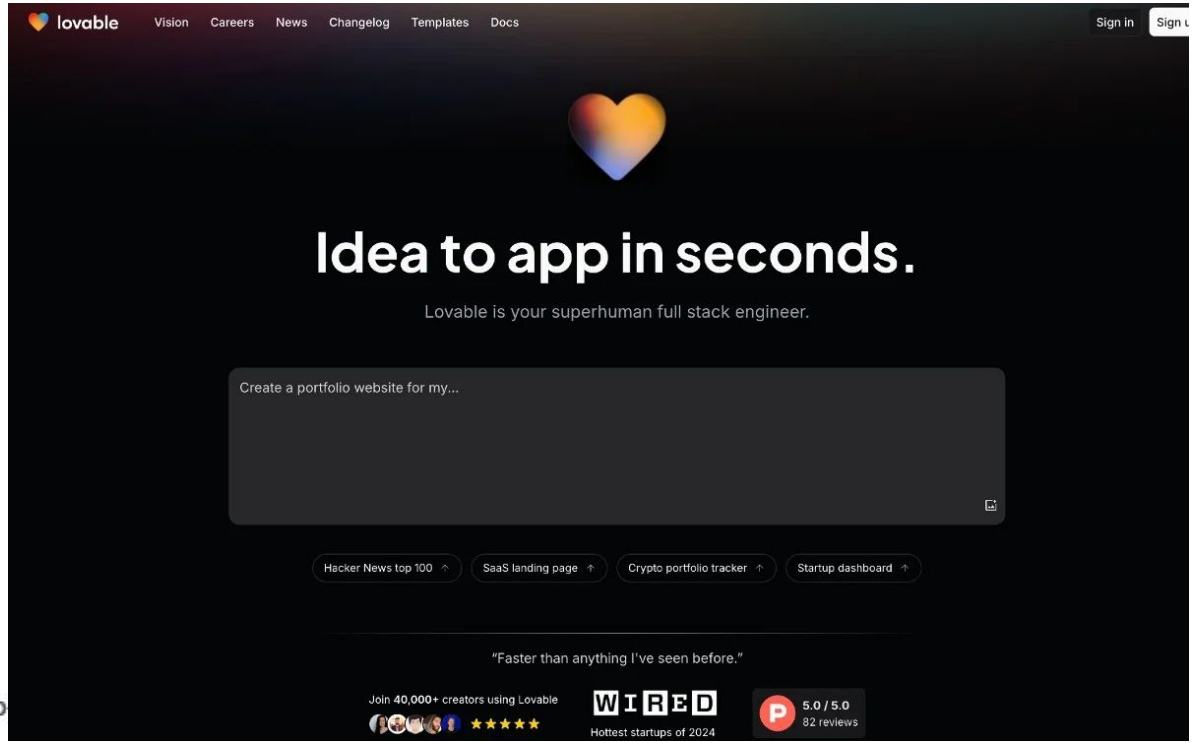
- Claude Code
- Codex (2025) [can run headless in CI!]
- Google Jules
- Devin
- Aider

Background / cloud run mode is an interesting use case



No-code / low-code

- Loveable
- v0.dev



Lovable
https://lovable.dev › blog › 2025-01-29-zero-to

Zero to $10M ARR in 2 months

My current preference: Embedded AI agents in IDEs

I currently like the form factor of the Cursor class of tools since:

- The Cursor agent can primarily drive changes
  - Has knowledge of your codebase:
    - No need to copy/paste to/from ChatGPT, etc.
    - Follows general conventions / patterns
  - Has Copilot-like autocomplete built in
  - Software engineer can edit inline
- VS Code fork
  - So you get extensions like linters, syntax highlighting, remote pair programming, etc.
  - The keyboard shortcuts are the same, etc.
- You can try different models since it's not owned by OpenAI / Anthropic, etc.
- Pricing

What's an agent?

Simplest definition (open to others):

"Agents are models using tools in a loop"

Most of these agents will have tools like file editing, codebase search, and more

They can use terminal commands and inspect those as well

Can use other tools via MCP or APIs

How does Anthony use Cursor?

Basic algorithm

- Try solving problems with the Cursor agent
  - Provide context / references as needed
  - Use a good model
- Use .cursor/rules files to encode your preferences
- Fix it when it goes off the rails
  - When the agent runs into issues, have it write a new rule about it and apply it to fix that going forward
- Use a "requirements" document to manage complex tasks

Rules example

The agent would be running a terminal command

I would terminate it since I saw an early error.

However, then the agent would just assume that the command had completed successfully!

I had to keep telling it to read the instructions more carefully…

.cursor/rules/keyboard-interrupt.mdc

```
---
description: How to handle KeyboardInterrupt in terminal command output
globs: "*"
alwaysApply: true
---
# Keyboard interrupt Handling

I will sometimes terminate a terminal / console command.

When you observe a KeyboardInterrupt (Ctrl+C, ^C, SIGINT) in command output:

1. Treat it as an indication that the command may have failed or had issues.
2. Do not assume the command completed successfully.
3. Inspect the terminal output closely.
4. Consider the need to diagnose and fix any underlying issues that led to the
   interruption.
5. Propose improvements to prevent the need for manual interruption.
```

Running and fixing RSpec tests

````
---
description: Run RSpec Tests After Editing
globs:
alwaysApply: true
---
# Run RSpec Tests After Edit

When an RSpec file (`spec/**/*_spec.rb`) is created or edited, run the tests
within that file to verify the changes and catch any regressions or errors.

Use a command like:

```sh
# Run all tests in the file
./bin/rspec path/to/your_spec.rb

# Run a specific test by line number (useful after adding/modifying one test)
./bin/rspec path/to/your_spec.rb:line_number
```

Debug and fix any failures that arise before proceeding.
````

Local rules

````
---
description: Rails schema table command
globs:
alwaysApply: true
---
On this computer, you may use the custom `table` CLI command to more easily
inspect the schema of the Rails application (which may be large).

Example:

```bash
$ table users
 create_table "users" do |t|
   t.string "name"
   t.string "email"
   t.string "password_digest"
   t.datetime "created_at", null: false
   t.datetime "updated_at", null: false
 end
```

Note that this will also include things like indexes, etc.
````

The meta-rule: How to write new rules

```
---
description: Adding or editing Cursor rules (to encode
"always" doing something)
globs:
alwaysApply: true
---
# Rules (meta)

Rule rules:

 - To add a new rule, create a file in .cursor/rules with .mdc
extension
 - Use kebab-case for rule filenames
 - Do not use leading numbers for rule filenames

Good rule filename examples:

 - .cursor/rules/my-new-rule.mdc
 - .cursor/rules/some-folder/my-new-rule.mdc  (it's ok to have
subfolders for organization)

Bad rule filename examples (with reason for being
problematic):

 - .cursor/rules/00-my-new-rule.mdc  (has leading numbers)
 - .cursor/rules/my_new_rule.mdc     (uses underscores instead
of dashes)
 - .cursorrules                      (not under .cursor/rules)

All rule files must have the following frontmatter:

```
---
description: <Several word text description of the rule>
globs: <any file globs that this should apply to>
alwaysApply: <true if global, otherwise false>
---
...then the rest of the file is standard Markdown...
```

If the user prompts you to add a rule, follow the process above.
If the user says to "always" do something, add a new rule for it.
If you make the same mistake more than once, add a new rule for it.

See [markdown.mdc](mdc:.cursor/rules/markdown.mdc) for rules around
Markdown formatting.
```

Handling complex features

How can we develop big features or do large scale changes?

Challenge: the agent / LLM can only handle so much context …

Concept: Context Windows (tokens)

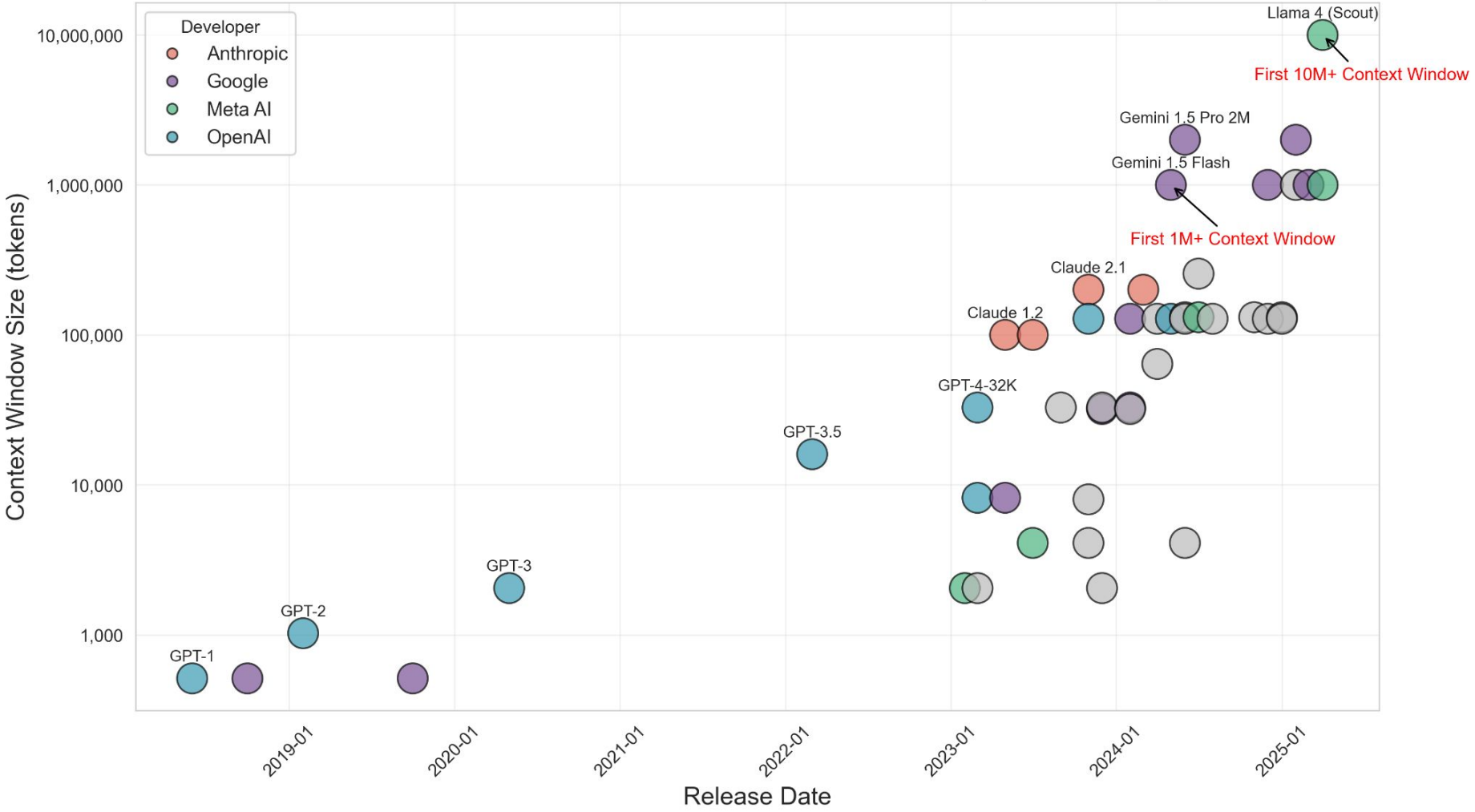
Tokens	Characters
64	252

Many words map to one token, but some don't: indivisible.

Unicode characters like emojis may be split into many toke
the underlying bytes: ������

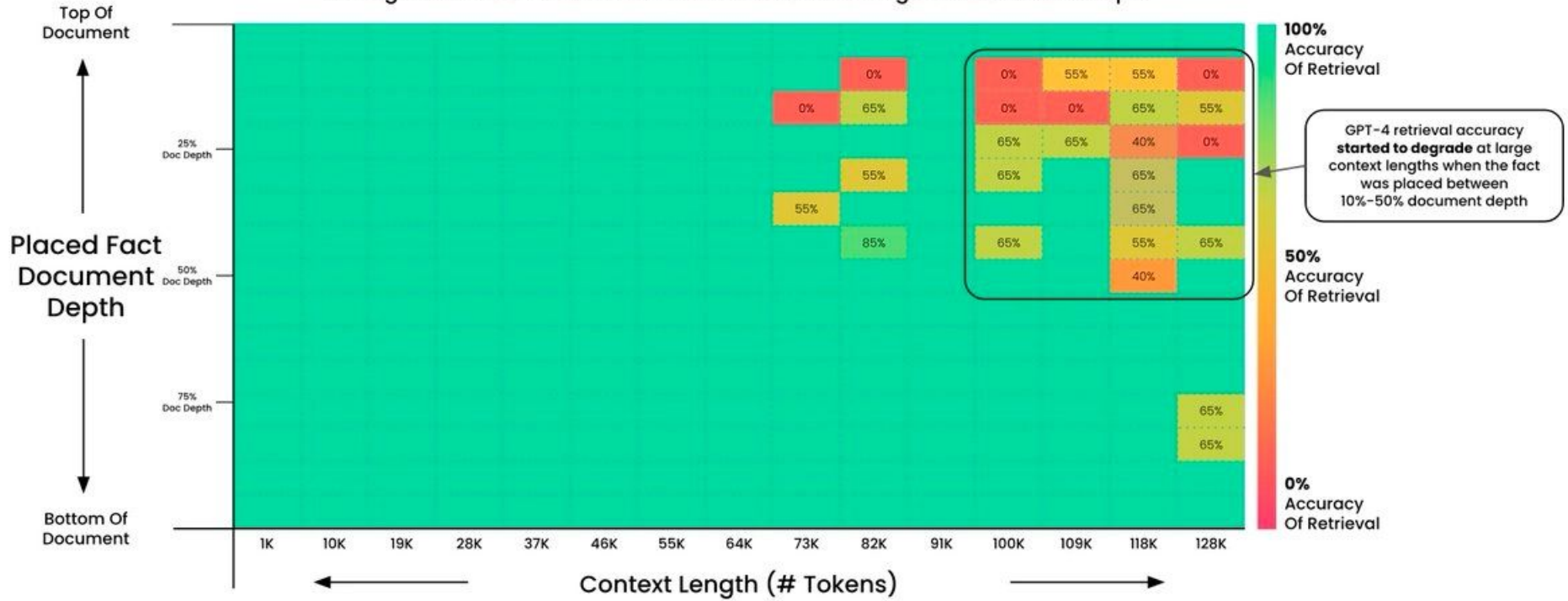
Sequences of characters commonly found next to each other
together: 1234567890

TEXT TOKEN IDS


Evolution of LLM Context Window Sizes (2018-2025)
Developer
Anthropic
Google
Meta AI
OpenAI
Context Window Size (tokens)
10,000,000
1,000,000
100,000
10,000
1,000
Release Date
2019-01
2020-01
2021-01
2022-01
2023-01
2024-01
2025-01
Llama 4 (Scout)
First 10M+ Context Window
Gemini 1.5 Pro 2M
Gemini 1.5 Flash
First 1M+ Context Window
Claude 2.1
Claude 1.2
GPT-4-32K
GPT-3.5
GPT-3
GPT-2
GPT-1

Pressure Testing GPT-4 128K via "Needle In A HayStack"

Asking GPT-4 To Do Fact Retrieval Across Context Lengths & Document Depth



Goal: Test GPT-4 Ability To Retrieve Information From Large Context Windows

A fact was placed within a document. GPT-4 (1106-preview) was then asked to retrieve it. The output was evaluated for accuracy.
This test was run at 15 different document depths (top > bottom) and 15 different context lengths (1K >128K tokens).
2x tests were run for larger contexts for a larger sample size.



Simon Willison
@simonwillison.net

Workaccount2 on Hacker News just coined the term "context rot" to describe the thing where the quality of an LLM conversation drops as the context fills up with accumulated distractions and dead ends news.ycombinator.com/item?id=4430...

> Workaccount2 9 hours ago | parent | next [–]
>
> They poison their own context. Maybe you can call it context rot, where as context grows and especially if it grows with lots of distractions and dead ends, the output quality falls off rapidly. Even with good context the rot will start to become apparent around 100k tokens (with Gemini 2.5).
>
> They really need to figure out a way to delete or "forget" prior context, so the user or even the model can go back and prune poisonous tokens.
>
> Right now I work around it by regularly making summaries of instances, and then spinning up a new instance with fresh context and feed in the summary of the previous instance.

ALT

Handling complex features

For any feature / bug / refactoring / upgrade that is likely more than one or two prompts away…

Specify it in English and then ask the agent to write a new file in requirements/*.md for it

You can see the plan, change it, remove unneeded things, ask for tests, point to resources, etc.

Continually reference the requirements file and ask it to implement the next feature, updating the requirements file and committing as you go

If the context window is filling up, then ask for it to update the requirements file and start a new chat

I prefer encoding the requirements doc as automated tests and then eventually removing

.cursor/rules/requirements-maintenance.mdc

```
description: Creating or editing requirements files
globs:
alwaysApply: true
---
# Requirements Maintenance Rules

Requirements hold the high-level requirements for the project.

All features must be added to a well-named file in the `requirements` folder

## Adding New Features

- Each feature must be described in detail, including its purpose and requirements
- Use subfolders of `requirements` to organize related features
- Use GitHub-style checkboxes `- [ ]` for all features
- Place new features in the most appropriate file
- If no appropriate file exists, create a new one
- Put a descriptive comment at the top of each file (e.g. "# User Authentication Features")

## Implementing Features

- Update checkbox to `- [x]` when feature is fully implemented
- Add implementation details as sub-items if needed
- Keep requirements synchronized with actual implementation

## Validation

- Verify requirements match implemented features
```

General recommendations

- Work iteratively
- Commit after working chunks
- Have the agent write rules for things that it didn't get or that you had to help with
- Have the agent run as many commands as it can, debugging as needed
- Use good models

Give fast, unambiguous feedback about the system

Vibes < Rules < (Linters + Tests + Types)

Then part of our (new) job is to:

- Enable linters, formatters to fix more issues automatically
- Provide tools for the agent to use to see and fix problems
- Provide context and guide the agent
- Provide permission and structure ("add new test scripts in tmp/ai/scripts")
- Add debug flags to give the agent more context
- Trimming unnecessary output to reduce unnecessary context doesn't fill up
- Clear error messages and exit codes

What's a "good model"?

- Using a good model is key to solving problems quickly and effectively
  - If the agent is spinning or just not getting it, try a different / better model
- It might be more expensive, but generally worth it
- Caveat: The same model might give you different performance on different days or different versions
- Check https://aider.chat/docs/leaderboards/ for top models

Aider polyglot coding leaderboard

Search...

View | Select | D

Model	Percent correct	Cost	Command	Correct edit format
gemini-2.5-pro-preview-06-05 (32k think)	83.1%	$49.88	`aider --model gemini/gemini-2.5-pro-preview-06-05 --thinking-tokens 32k`	99.6%
o3 (high) + gpt-4.1	82.7%	$69.29	`aider --model o3 --architect`	100.0%
o3 (high)	79.6%	$111.03	`aider --model o3`	95.1%
gemini-2.5-pro-preview-06-05 (default think)	79.1%	$45.6	`aider --model gemini/gemini-2.5-pro-preview-06-05`	100.0%
Gemini 2.5 Pro Preview 05-06	76.9%	$37.41	`aider --model gemini/gemini-2.5-pro-preview-05-06`	97.3%
Gemini 2.5 Pro Preview 03-25	72.9%		`aider --model gemini/gemini-2.5-pro-preview-03-25`	92.4%
claude-opus-4-20250514 (32k thinking)	72.0%	$65.75	`aider --model claude-opus-4-20250514`	97.3%
o4-mini (high)	72.0%	$19.64	`aider --model o4-mini`	90.7%

Things I've actually used the agent for

Things I've actually used this for

Writing one-off reporting / analysis scripts

- Easy to spot-check / verify, the model can run it and debug
- Low risk in my context

Adding real features

Fixing real bugs on real applications

- Provide the bug description and any context you can think of, and then it might be able to fix or at least see where the issue is
- And it will write regression tests to prevent it from happening in the future!

Things I've actually used this for

Adding or updating CI scripts (GitHub Actions)

- Running license_checker
- Running linters with caching
- Running tests in parallel

Small maintenance things

- Fixing peer dependency warnings

Things I've actually used this for

Upgrading a Rails app from 7.0 -> 7.1, 7.1 -> 7.2, etc.

- It will patiently work through new config settings
- It can analyze the codebase for problems
- It can run the tests and fix new failures
- It can clean up deprecation warnings
- It can make small commits after each new fix

Things I've

```
Checking your local Rails version... 7.1.5.1
Fetching EOL data from https://endoflife.date/api/rails.json ...
Fetching release data from https://rubygems.org/gems/rails/versions ...

--- Rails 8.0 Releases --- (EOL: 2026-11-07)
  8.0.2              (Released: 2025-03-12)  <-- Latest
  8.0.1              (Released: 2024-12-13)
  8.0.0.1            (Released: 2024-12-10)
  8.0.0              (Released: 2024-11-07)

--- Rails 7.2 Releases --- (EOL: 2026-08-09)
  7.2.2.1            (Released: 2024-12-10)  <-- Latest
  7.2.2              (Released: 2024-10-31)
  7.2.1.2            (Released: 2024-10-23)
  7.2.1.1            (Released: 2024-10-15)
  7.2.1              (Released: 2024-08-22)
  7.2.0              (Released: 2024-08-09)

--- Rails 7.1 Releases --- (EOL: 2025-10-01)
  7.1.5.1            (Released: 2024-12-10)  <-- You are here
  7.1.5              (Released: 2024-10-31)
  7.1.4.2            (Released: 2024-10-23)
  7.1.4.1            (Released: 2024-10-15)
  7.1.4              (Released: 2024-08-22)
  ...
  7.1.3.1            (Released: 2024-02-21)
  7.1.3              (Released: 2024-01-16)
  7.1.2              (Released: 2023-11-10)
  7.1.1              (Released: 2023-10-11)
  7.1.0              (Released: 2023-10-05)
```

Common Challenges

Agent wants you to run a script / the tests

"No, you run it, and keep running and fixing until done"

Add a rule around it

The agent just goes off the rails

- Often this is related to the context being full or wrong or both
- Stop execution, fix misconceptions or give more guidance
- Reset to the last known good checkpoint / commit and try again
- Start a new chat
  - You'll need to do this if you see it timing out or going in logic/fix loops
- Start smaller or work more iteratively
- Do we need to simplify?
  - Kent Beck: "Make the change easy (warning: this may be hard), then make the easy change"
  - "If it's confusing/hard for humans, it's more likely confusing/hard for the model"

Agent is using the wrong version of a library

Example: Agent is using React v14 when it should be using v15 (or vice versa)

This is just whatever is in its training set or a guess

Add correct docs using Cursor's docs feature

Add rules

Sometimes this can burn a lot of time

- Next.js versions have very different directory structures

If time permits, add a feature

Future / speculative

- Adding context for Rails / server logs
- Background agents
  - Try to open a PR for every Sentry/Rollbar/Honeybadger error that comes in
  - Takes a look at flaky tests and tries to fix them, running test 1000 times to verify fixes
- Could use git-worktree to split off separate codebase versions locally
  - One challenge is getting a truly separated environment for test runs, etc.
- New bottleneck: review and testing
  - Now I have 1000 line PR that seems to be working. But is it?
  - In my experience, AI-assisted code review probably makes it worse
  - `mostly AI` review label
- Process questions:
  - Should developers self-review AI-generated code? Or get another person to review?
  - Should we commit .cursor/rules or requirements documents?