DEEPDIVES

How Claude Code is built

A rare look into how the new, popular dev tool is built, and what it might mean for the future of software building with AI. Exclusive.

GERGELY OROSZ
SEP 23, 2025 · PAID

425 13 34 Share ···

Claude Code has taken the developer world by storm since being made generally available in May. The tool is currently generating more than $500M in annual run-rate revenue, and usage has exploded by more than 10x in the three months since that May release.

I recently sat down with two of the founding engineers behind Claude Code: Boris Cherny (the engineer who came up with the original prototype, and the founding engineer of the project), Sid Bidasaria (engineer #2 of Claude Code, and creator of Claude Code subagents), and founding product manager Cat Wu.

I learned how Claude Code is built, and got insights into how a successful "AI-first engineering team" operates; it was a bit like a peek into a crystal ball and a potential future of how fast-moving startups will operate. The good news is that software engineers appear *very much* in demand in it…

Today, we cover:

1. **How it all began.** The idea for Claude Code came from a command-line tool using Claude to state what music an engineer was listening to at work. It spread like wildfire at Anthropic after being given access to the filesystem. Today, Claude Code has its own fully-fledged team.
2. **Tech stack and architecture.** TypeScript, React, Ink, Yoga, and Bun. The tech stack was chosen to be "on distribution" and to play to the strengths of the model.

Fun fact: 90% of code in Claude Code is written by itself!

3. **Building and shipping features in days – not weeks**. The team is working at rapid pace, with around 5 releases per engineer each day. Prototyping is done surprisingly quickly: we go through 10+ actual prototypes for a new feature. *It looks like AI agents really speed up iteration.*
4. **Redesigning the terminal UX.** Claude Code brings a lot of innovative features to the terminal user experience that weren't needed before we could interact with terminals powered by LLMs. A look at some of them.
5. **What "AI-first" software engineering looks like.** Using AI agents for code reviews and tests, test-driven development's (TDD) renaissance, automating incident response, and cautious use of feature flags. Does this presage how "AI-first" engineering teams will work in the future?
6. **Building subagents.** A walk through how the subagents feature was built in just three days, of which two days' work was thrown away.
7. **Future of AI-assisted engineering teams?** What engineering teams can learn from how Anthropic operates, and things to keep in mind about what makes it unique, that might be less transferable.

As a note, working in tech, you need to be living under a rock not to have heard about Claude Code. That is, unless you only pay attention to Gartner: in the latest report on AI coding assistants, Gartner does not even mention Claude Code (and gets plenty of other things very wrong.) One more reason to subscribe to The Pragmatic Engineer to keep up with the tech industry — or risk being left months or years behind with Gartner.

Other, related deepdives from the real-world engineering challenges series:

- *Real-world engineering challenges: building Cursor*
- *How Anthropic built Artifacts*
- *Scaling ChatGPT: five real-world engineering challenges*

1. How it all began

Boris Cherny joined Anthropic in September 2024, and began building a bunch of different prototypes with the Claude 3.6 model. At the time, he wanted to get more familiar with Anthropic's public API. Boris recalls that period:

> "I started hacking around using Claude in the terminal. The first version was barebones: it couldn't read files, nor could it use bash, and couldn't do any engineering stuff at all. But it could interact with the computer.
>
> **I hooked up this prototype to AppleScript**: it could tell me what music I was listening to while working. And then it could also change the music playing, based on my input.
>
> This was a cool demo, but wasn't that interesting".

Meanwhile, Cat Wu was researching the computer usage of AI agents, and new capabilities that arose from agents using them. Following a conversation with Cat, Boris had the idea to give the terminal more capabilities than just using AppleScript. He says:

> "I tried giving it some tools to interact with the filesystem and to interact with the batch; it could read files, write files, and run batch commands.
>
> Suddenly, this agent was *really* interesting. I ran it in our codebase, and just started asking questions about it. Claude then started exploring the filesystem and reading files. So, it would read one file, look at the imports, then read the files that were defined in the imports! It went on, until it found a good answer. Claude exploring the filesystem was mindblowing to me because I'd never used any tool like this before.
>
> **In AI, we talk about "product overhang", and this is what we discovered with the prototype.** Product overhang means that a model is able to do a specific thing, but the product that the AI runs in isn't built in a way that captures this capability. What I discovered about Claude exploring the filesystem was pure product

> overhang. The model could already do this, but there wasn't a product built around this capability!"

Product-market fit

Boris started to use his prototype every day at work. He then shared it with what would become the core Claude Code team, and fellow devs started to use it daily, too.

Boris and the Claude Code team released a dogfooding-ready version in November 2024 – two months after the first prototype. On the first day, around 20% of the Engineering team used it, and by day five, 50% of Engineering was using Claude Code. At that point, Boris felt pretty confident Claude Code could be a hit in the outside world.

But there was internal debate about whether to even release the tool, or to keep it for internal use. Boris recounts:

> "We actually weren't even sure if we wanted to launch Claude Code publicly because we were thinking it could be a competitive advantage for us, like our "secret sauce": if it gives us an advantage, why launch it?
>
> In the end, this is the position we landed on:
>
> - Anthropic, at its core, is a model safety company
> - The way we learn about model safety and capabilities is that we make tools people use
> - Claude Code would *almost certainly* be a tool people use because all of Anthropic got hooked on it
>
> So by releasing this tool, we learn a lot more about model safety and capabilities."

Assembling the Claude Code engineering team

Initially, the team was just Boris until in November, Sid Bidasaria joined Anthropic and came across the early version of Claude Code. He liked the idea and joined Boris on the project.

There was a lot of freedom in how their two-person team worked. Sid told me:

> “Most of what we did was prototype *really* quickly and build products that showcase how strong the underlying model is. We didn’t have formal processes inside the team: it was all super fluid. We could work on pretty much whatever we wanted, and so we just kept choosing the most promising ideas."

The team grew to around 10 engineers by July, and hiring has continued since. Today, it’s a full-fledged product team with engineers, Product Management, Design, and Data Science folks – and they’re still hiring.

Claude Code not only for coders

Today, more than 80% of Anthropic’s engineers who write code use Claude Code day-to-day, but it’s not only them. Boris:

> “I was walking into the office and glanced at the screen of a data scientist. He had Claude Code running. I was like, “hold on, why do you have Claude Code running?” He says: “I figured out how to get this thing running and write queries for me.”
>
> These days when I walk by the row of data scientists, they all have Claude Code running – many of them have several instances– running queries, creating visualizations, and doing other types of helpful work”.

An interesting point is that Boris only ever had software engineers in mind for Claude Code – hence the name! – but the product has shown it has further utility in other areas.

More engineering output while doubling team size

If we take the metric of pull requests per engineer; when an engineering team doubles in size quickly this process usually pulls down the metric. This is because existing engineers spend more time onboarding new colleagues and less time coding, and new joiners need to get the hang of things at first. *Like any metric, pull requests (PR) per engineer are not a perfect metric, but it does give a sense of the pace of iteration. PR throughput is [measured by companies](#) including GitHub, Dropbox, Monzo, Adyen, and others.*

But Anthropic saw a 67% increase in PR throughput as their team size doubled – thanks to Claude Code. It would have been normal for the average-PRs-merged metric to drop, but it actually went up! The credit for this is given to Claude Code: engineers get PRs done faster with it. In what might have been a lucky constellation of events, Anthropic doubled its engineering headcount at around the time that Claude Code was adopted across all of engineering.

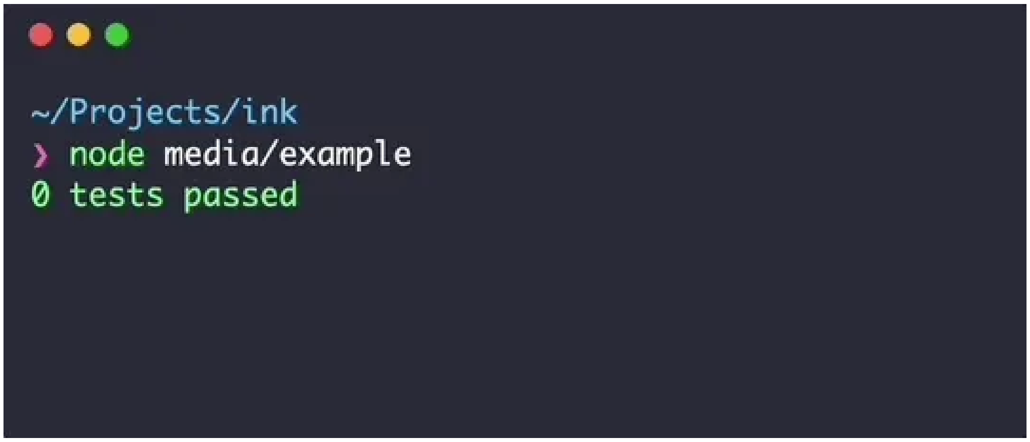
I've also noticed that finishing a piece of work with Claude Code is considerably faster, and that I make better progress on my coding tasks with the tool. It also helps when I build stuff for which Claude Code can verify the correctness of the code, by running the program and checking outputs or running tests.

2. Tech stack and architecture

Claude Code's tech stack:

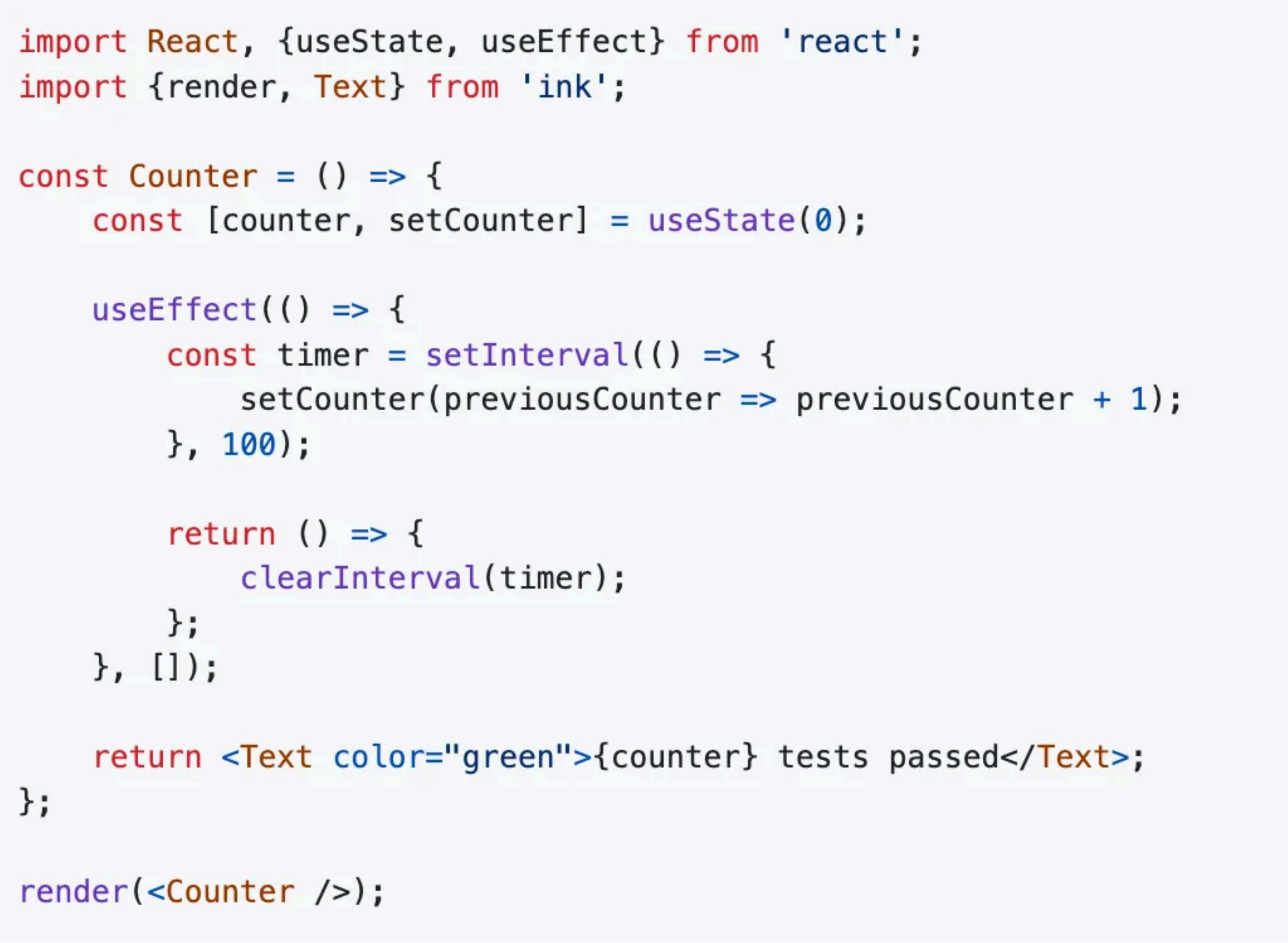
- **TypeScript**: Claude Code is built on this language
- **React with Ink:** the UI is written in React, using the Ink framework for interactive command-line elements
- **Yoga:** the layout system, open sourced by Meta. It's a constraints-based layout that works nicely. Terminal-based applications have the disadvantage of needing to support all sizes of terminals, so you need a layout system to do this pragmatically
- **Bun**: for building and packaging. The team chose it for speed compared to other build systems like Webpack, Vite, and others.

The Ink framework is a neat component that allows for creating pleasant-looking UIs in the terminal. For example, to create this UI:



```
~/Projects/ink
❯ node media/example
0 tests passed
```

You can write the code in React:



```
import React, {useState, useEffect} from 'react';
import {render, Text} from 'ink';

const Counter = () => {
    const [counter, setCounter] = useState(0);

    useEffect(() => {
        const timer = setInterval(() => {
            setCounter(previousCounter => previousCounter + 1);
        }, 100);

        return () => {
            clearInterval(timer);
        };
    }, []);

    return <Text color="green">{counter} tests passed</Text>;
};

render(<Counter />);
```

The code for a counter that updates in the console. Source: Ink on GitHub

npm is used for distributing Claude Code. It’s the most popular package manager in the Node ecosystem. To get started with Claude Code, you need to have Node 18-or-above installed, then run:

npm install -g @anthropic-ai/claude-code

Once that's done, you can start the tool with the *Claude* command.

The tech stack was chosen to be "on distribution" for the Claude model. In AI, there are the terms "on distribution" and "off distribution." "On distribution" means the model already knows how to do it, and "off distribution" means it's not good at it.

The team wanted an "on distribution" tech stack for Claude that it was already good at. TypeScript and React are two technologies the model is very capable with, so were a logical choice. However, if the team had chosen a more exotic stack Claude isn't that great with, then it would be an "off distribution" stack. Boris sums it up:

> "With an off-distribution stack, the model can still learn it. But you have to show it the ropes and put in the work. We wanted a tech stack which we didn't need to teach: one where Claude Code could build itself. And it's working great; around 90% of Claude Code is written with Claude Code".

Architecture: choose the simplest option

Interestingly, there's not all that much to Claude Code in terms of modules, components, and complex business logic on the client side. For a tool that does pretty complex things like traversing filesystems and codebases, this is somewhat surprising! But Claude Code is just a lightweight shell on top of the Claude model. This is because the model does almost all of the work:

- Defines the UI, and exposes hooks for the model to modify it
- Exposes tools for the models to use
- … then gets out of the way

The Claude Code team tries to write as little business logic as possible. Boris tells me:

> “This might sound weird, but the way we build this is we want people to *feel* the model as *raw* as possible. We have this belief the model can do much more than products today enable it to do.
>
> When you look at a lot of coding products, they get in the way of the model; they add scaffolding by adding UI elements and other parts that clutter things, so that the model running in those tools feels like it’s hobbling on one foot. Features that are meant to be helpful for users end up limiting the model. So, we try to make the UI as minimal as possible.
>
> **Every time there’s a new model release, we delete a bunch of code.** For example, with the 4.0 models, we deleted around half the system prompt because we no longer needed it. We try to put as little code as possible around the model, and this includes minimizing prompting and minimizing the number of tools. We constantly delete tools and experiment with new ones”.

Claude Code does not use virtualization – it runs locally. A major design decision was whether to run Claude Code in a virtual machine – like on a Docker container, or in the cloud – and thereby create a sandbox environment. But the team decided to go with a version that runs locally because: simplicity! From Boris:

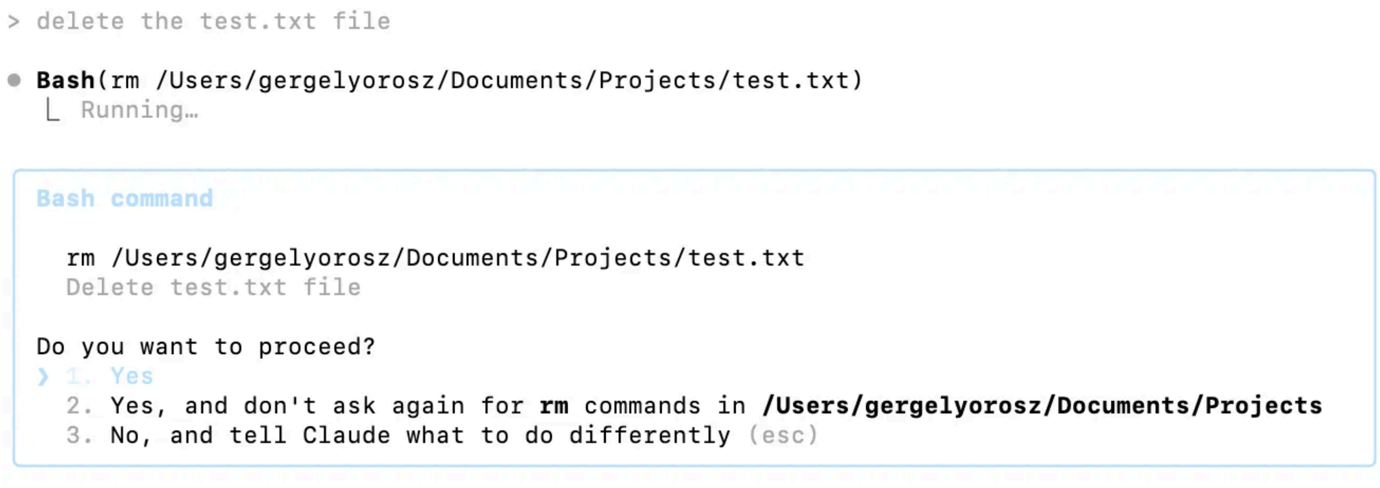
> “With every design decision, we almost always pick the simplest possible option. What are the simplest answers to the questions: “where do you run batch commands?” and “where do you read from the filesystem?” It’s to do it locally.
>
> So we went with this: Claude Code runs batch commands locally, and reads and writes to the filesystem. There’s no virtualization”.

The permissions system

The most complex part of Claude Code is the permissions system. The risk of running Claude Code locally is that an agent may do irreversible things that it shouldn’t, such as deleting files. But how can this be done securely?

Again, the team has opted for simplicity and built a permissions system that seeks permission before executing an action. The user can then decide to:

- Grant the permission once
- Grant the permission for future sessions, as well
- Reject the permission



```
> delete the test.txt file

● Bash(rm /Users/gergelyorosz/Documents/Projects/test.txt)
  ⎿ Running…

Bash command

  rm /Users/gergelyorosz/Documents/Projects/test.txt
  Delete test.txt file

Do you want to proceed?
❯ 1. Yes
  2. Yes, and don't ask again for rm commands in /Users/gergelyorosz/Documents/Projects
  3. No, and tell Claude what to do differently (esc)
```

Claude Code seeking permission before deleting a file

Boris tells me that getting permissions right took effort:

> “Our most important principle is this: if you start running Claude Code, it shouldn't change things on your system without permission. That could be dangerous.
>
> However, we should also give ways for people to opt out and say things like “actually, in the context I’m working in, I’d like to not give permission every time.”
>
> There’s a lot of nuance in the permissions system, though. For example, when a command comes in, we do static analysis on them to check if this is something already allowed in the settings file (in the settings.json file).
>
> **The settings system is a multi-tiered system that can be configured per project, per user, and per company.** You can also share settings with your team.

> We are observing teams share settings files that whitelists commands so that Claude Code won't ask for permission, such as checking into source control".

Other features

Claude Code is simple in some ways, but has dozens of features which add to its complexity. Several are documented. Some to note:

- Hooks: create custom shell commands for Claude Code to use
- MCP support: give Claude Code more capabilities by connecting it to MCP servers. See our deepdive on the Model Context Protocol (MCP)
- GitHub and GitLab support: use GitHub Actions and integrate Claude Code into GitLab CI/CD.
- Output styles: the ability to switch to output styles, or define your own. Built-in output styles you can switch to include:
  - Explanatory: educating you about implementation choices
  - Learning: a collaborative style where Claude asks you to do small tasks yourself. A very clever approach to stay in the loop, hands-on, and learn! It could be a great way for less experienced engineers to learn, or for those unfamiliar
- Configuration: use a variety of config files and settings to configure your terminal, model, status line, and many more
- Subagents: covered below
- Enterprise features: set up Identity and Access Management (IAM) to use Claude Code across an organization, and access org-wide analytics to track usage
- Claude Code SDK: build custom AI agents, using the agent harness that powers Claude Code
- ... and see this roundup of recent new features in Claude Code

3. Building and shipping features in days, not weeks

For a team of around a dozen engineers, they work *really* fast:

~60-100 internal releases/day. Any time an engineer makes a change to Claude Code, they release a new npm package internally. Everyone at Anthropic uses the internal version and the dev team gets rapid feedback.

Over the summer, engineers pushed around 5 pull requests per day – a much faster pace than at most tech companies where 1-2 pull requests per day are often the norm.

1 external release/day. Almost every day, a new version of the package is released as part of a deployment.

20 prototypes in 2 days: building todo lists

What surprises me about this development is that the team does a lot more prototyping with Claude Code than I'm used to seeing. As an example, Boris walked me through how he built around 20 prototypes of the new feature, todo lists, in a few hours over two days.

He was kind enough to share the actual prompts he used for the various iterations. After each one, Boris:

- Sometimes tweaked the result
- Played around with it
- If it felt good, shared it with colleagues for feedback
- When something felt off, he built a new prototype with a new prompt

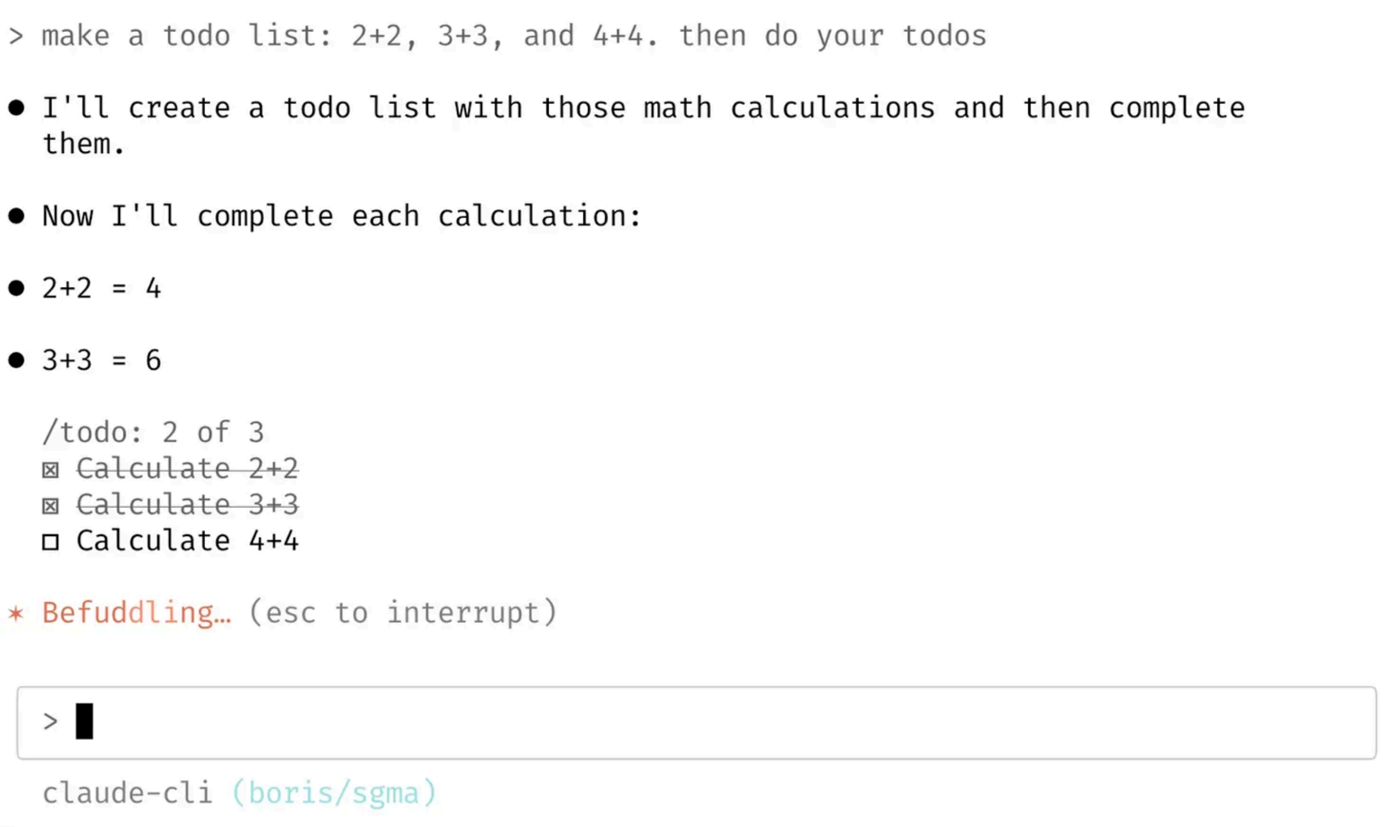
Prototype #1: showing todos as they are completed

Idea: todo lists are one of the easiest ways to keep track of how Claude is doing, so they tried having the list just below the most-recent tool call.

Prompt:

> > make it so instead of todos showing up as they come in, we hide the tool use and result for todos, and render a fixed todo list above the input. title it "/todo (1 of 3)" in grey

How it looked:



```
> make a todo list: 2+2, 3+3, and 4+4. then do your todos

● I'll create a todo list with those math calculations and then complete
  them.

● Now I'll complete each calculation:

● 2+2 = 4

● 3+3 = 6

  /todo: 2 of 3
  ☒ ~~Calculate 2+2~~
  ☒ ~~Calculate 3+3~~
  ☐ Calculate 4+4

✶ Befuddling… (esc to interrupt)

  > █

  claude-cli (boris/sgma)
```

Prototype #1

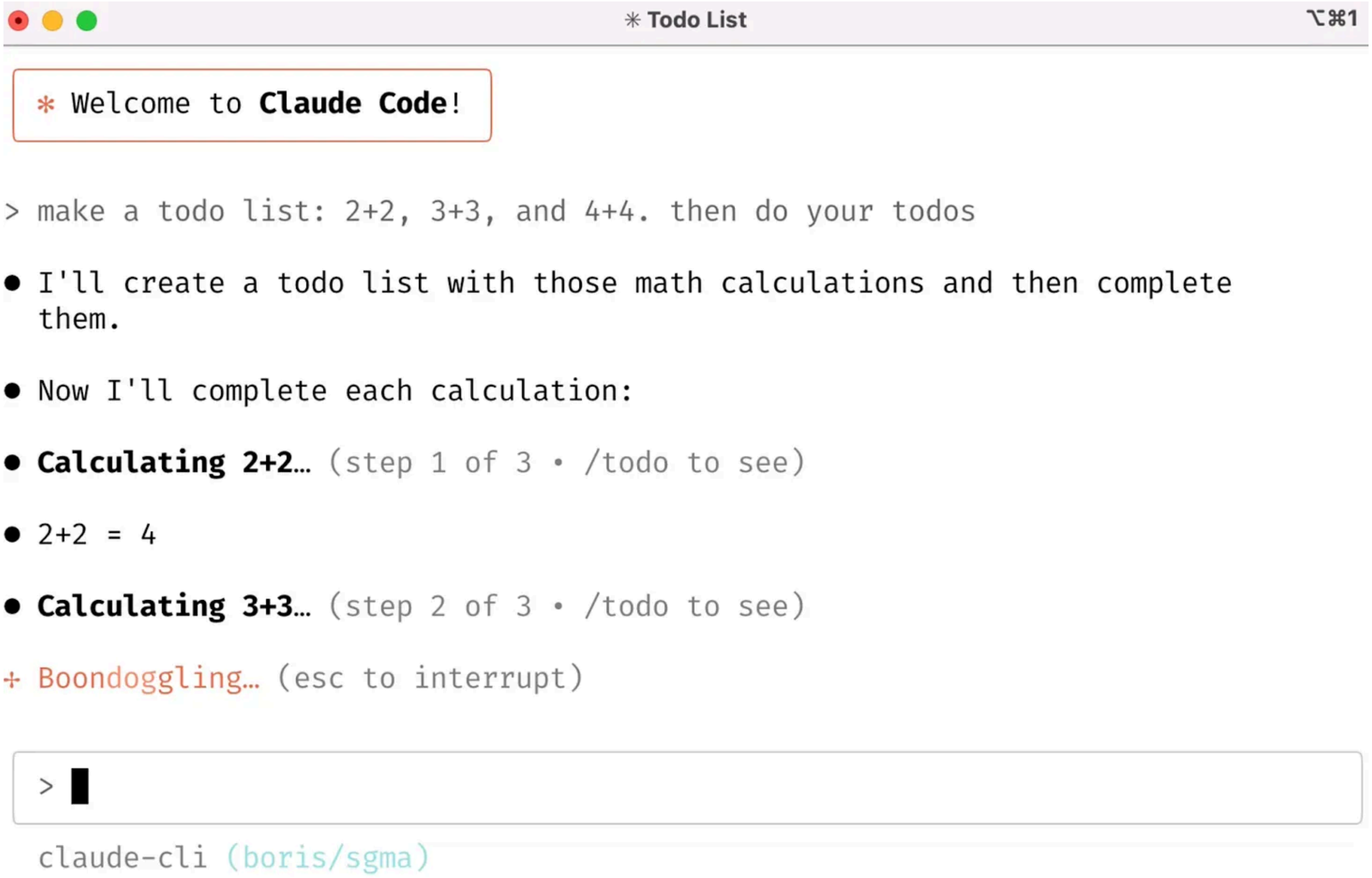
Prototype #2: showing progress at the bottom

Another variation was showing each todo update in line.

Prompt:

> > actually don't show a todo list at all, and instead render the tool used inline, as bold headings when the model starts working on a todo. keep the "step 2 of 4" or whatever, and add middot /todo to see after in grey

How it looked:



```
✻ Welcome to Claude Code!

> make a todo list: 2+2, 3+3, and 4+4. then do your todos

● I'll create a todo list with those math calculations and then complete
  them.

● Now I'll complete each calculation:

● Calculating 2+2… (step 1 of 3 • /todo to see)

● 2+2 = 4

● Calculating 3+3… (step 2 of 3 • /todo to see)

✢ Boondoggling… (esc to interrupt)

> █

  claude-cli (boris/sgma)
```

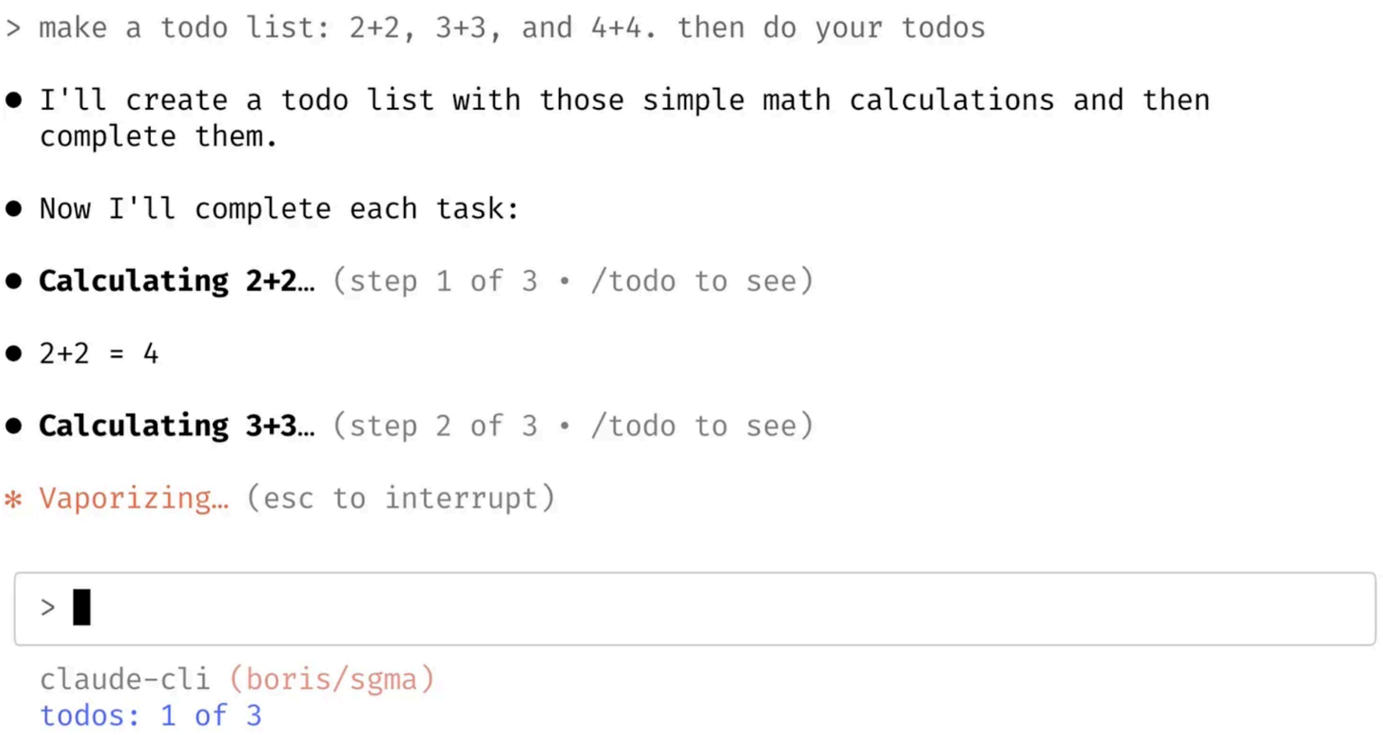
Prototype #2

Prototypes #3 and #4: an “interactive pill”

What if the todos were an interactive pill (a rectangle at the bottom of the console) that you could pull up to see progress on, just like background tasks?

Prompts and outputs:

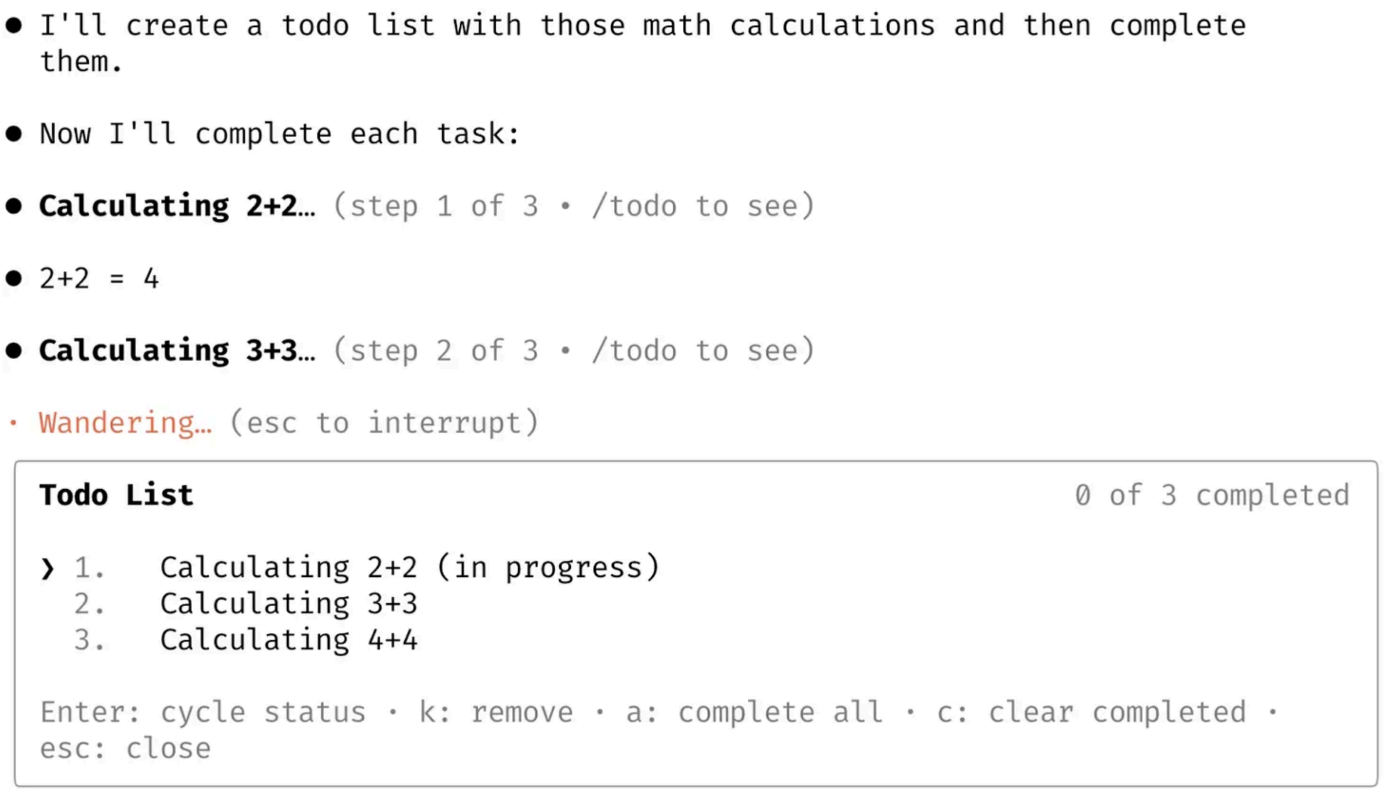
> > also add a todo pill under the text input, similar to bg tasks. it should render "todos: 1 of 3" or whatever



```
> make a todo list: 2+2, 3+3, and 4+4. then do your todos

● I'll create a todo list with those simple math calculations and then
  complete them.

● Now I'll complete each task:

● Calculating 2+2… (step 1 of 3 • /todo to see)

● 2+2 = 4

● Calculating 3+3… (step 2 of 3 • /todo to see)

* Vaporizing… (esc to interrupt)

 > ▌

  claude-cli (boris/sgma)
  todos: 1 of 3
```

Prototype #3

> \> make the pill interactive, like the bg tasks pill



```
● I'll create a todo list with those math calculations and then complete
  them.

● Now I'll complete each task:

● Calculating 2+2… (step 1 of 3 • /todo to see)

● 2+2 = 4

● Calculating 3+3… (step 2 of 3 • /todo to see)

· Wandering… (esc to interrupt)

  Todo List                                         0 of 3 completed

  ❯ 1.   Calculating 2+2 (in progress)
    2.   Calculating 3+3
    3.   Calculating 4+4

  Enter: cycle status · k: remove · a: complete all · c: clear completed ·
  esc: close
```

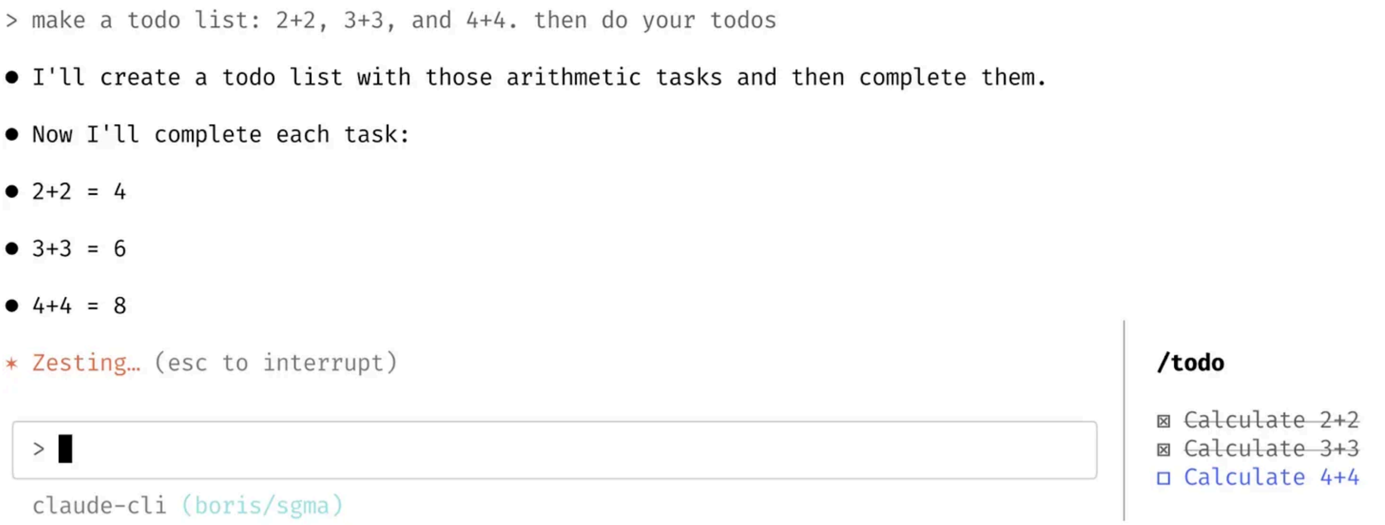
Prototype #4

Prototypes #5 and #6: using a “drawer”

What if we had a ‘drawer’ that slid in with the todos on the side?

Prompts and outputs:

> \> actually undo both the pill and headings. instead, make the todo list render to the right of the input, vertically centered with a grey divider. show it when todos are active, hide it when it's done



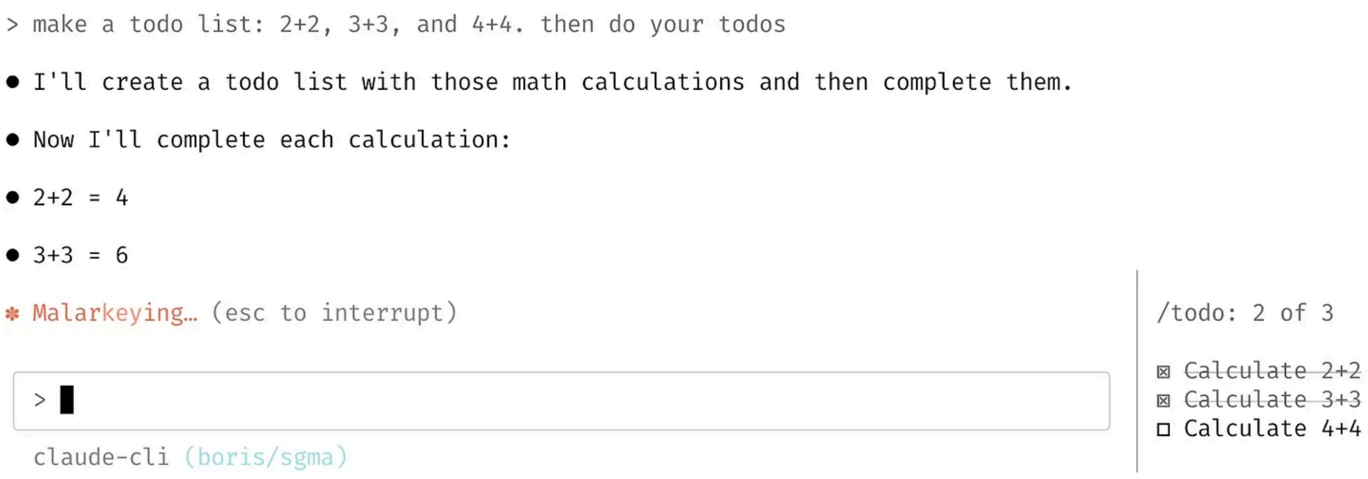
```
> make a todo list: 2+2, 3+3, and 4+4. then do your todos

● I'll create a todo list with those arithmetic tasks and then complete them.

● Now I'll complete each task:

● 2+2 = 4

● 3+3 = 6

● 4+4 = 8

✶ Zesting… (esc to interrupt)

 > █

 claude-cli (boris/sgma)
```

/todo

☒ ~~Calculate 2+2~~
☒ ~~Calculate 3+3~~
☐ Calculate 4+4

Prototype #5, with todos as a “drawer” on the right. See the animated version

> \> it's a little jumpy, can you also animate it like a drawer



```
> make a todo list: 2+2, 3+3, and 4+4. then do your todos

● I'll create a todo list with those math calculations and then complete them.

● Now I'll complete each calculation:

● 2+2 = 4

● 3+3 = 6

✻ Malarkeying… (esc to interrupt)                               /todo: 2 of 3

 ┌──────────────────────────────────────────────────┐          ☒ ~~Calculate 2+2~~
 │ > █                                              │          ☒ ~~Calculate 3+3~~
 └──────────────────────────────────────────────────┘          ☐ Calculate 4+4
  claude-cli (boris/sgma)
```

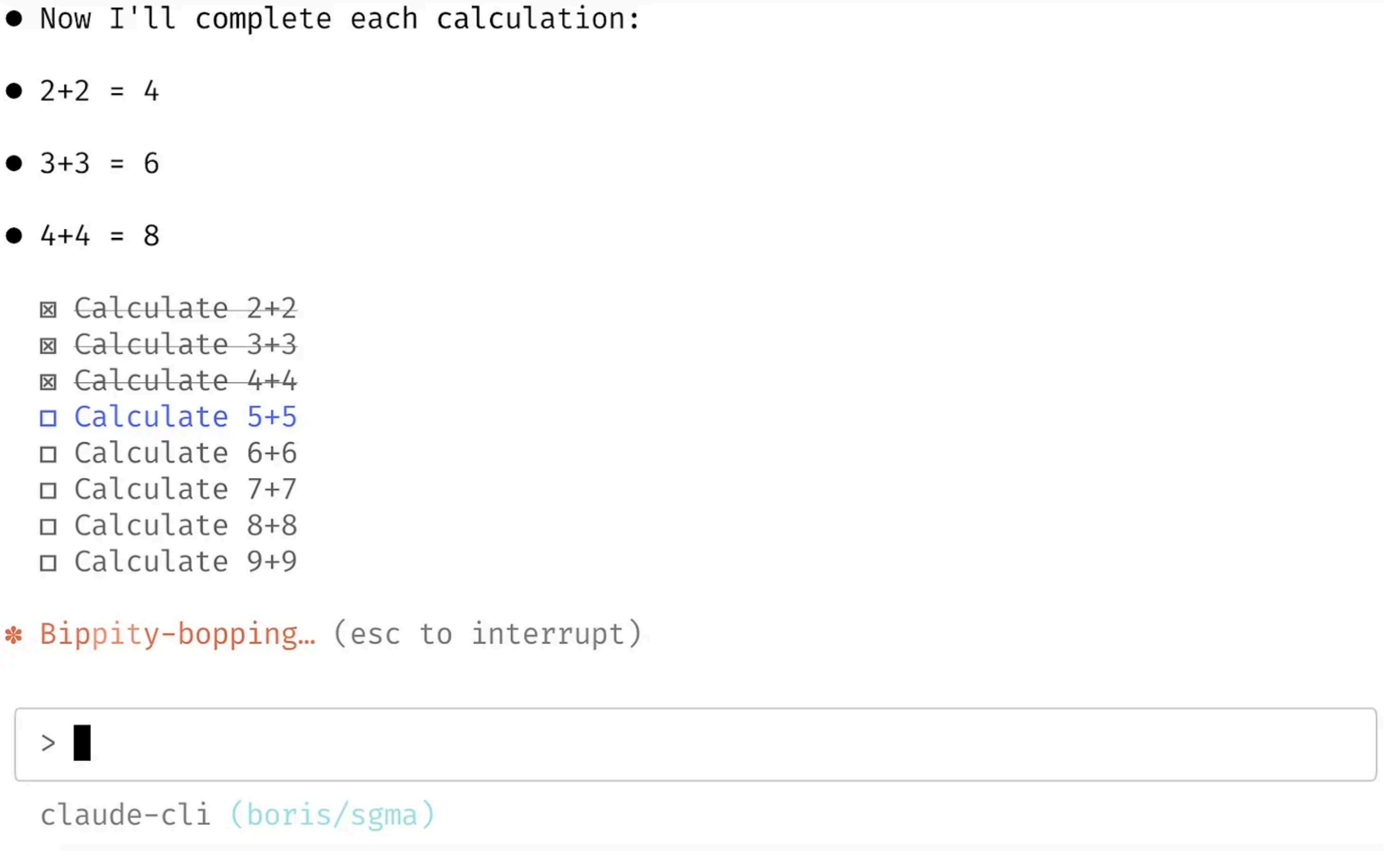
Prototype #6. See the animated version

Prototypes #7, #8 and #9: experiments on visibility

To make the todo list as visible as possible, Boris tried making it always show above the input.

Prompts and outputs:

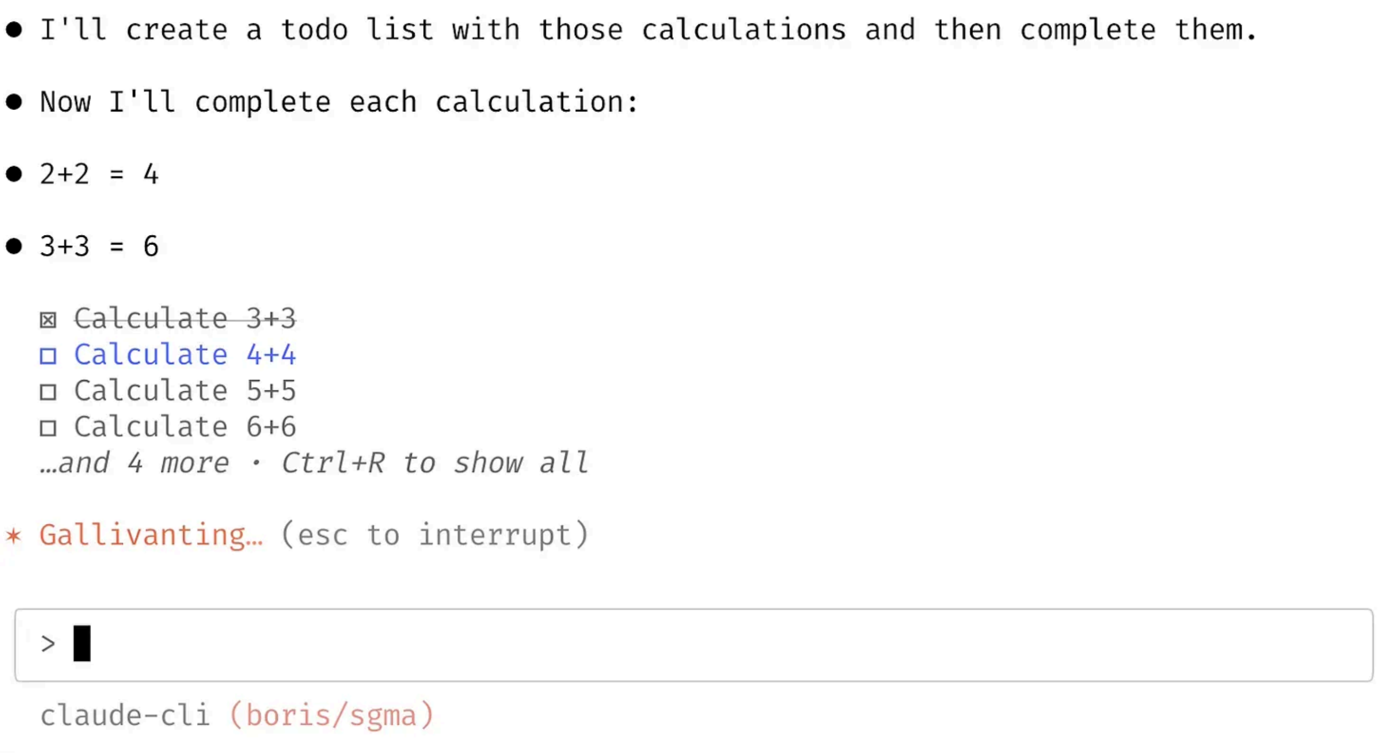
> > actually what if you show the todo list above the input instead



```
● Now I'll complete each calculation:

● 2+2 = 4

● 3+3 = 6

● 4+4 = 8

  ☒ ~~Calculate 2+2~~
  ☒ ~~Calculate 3+3~~
  ☒ ~~Calculate 4+4~~
  ☐ Calculate 5+5
  ☐ Calculate 6+6
  ☐ Calculate 7+7
  ☐ Calculate 8+8
  ☐ Calculate 9+9

✻ Bippity-bopping… (esc to interrupt)

 > █

  claude-cli (boris/sgma)
```

Prototype #7

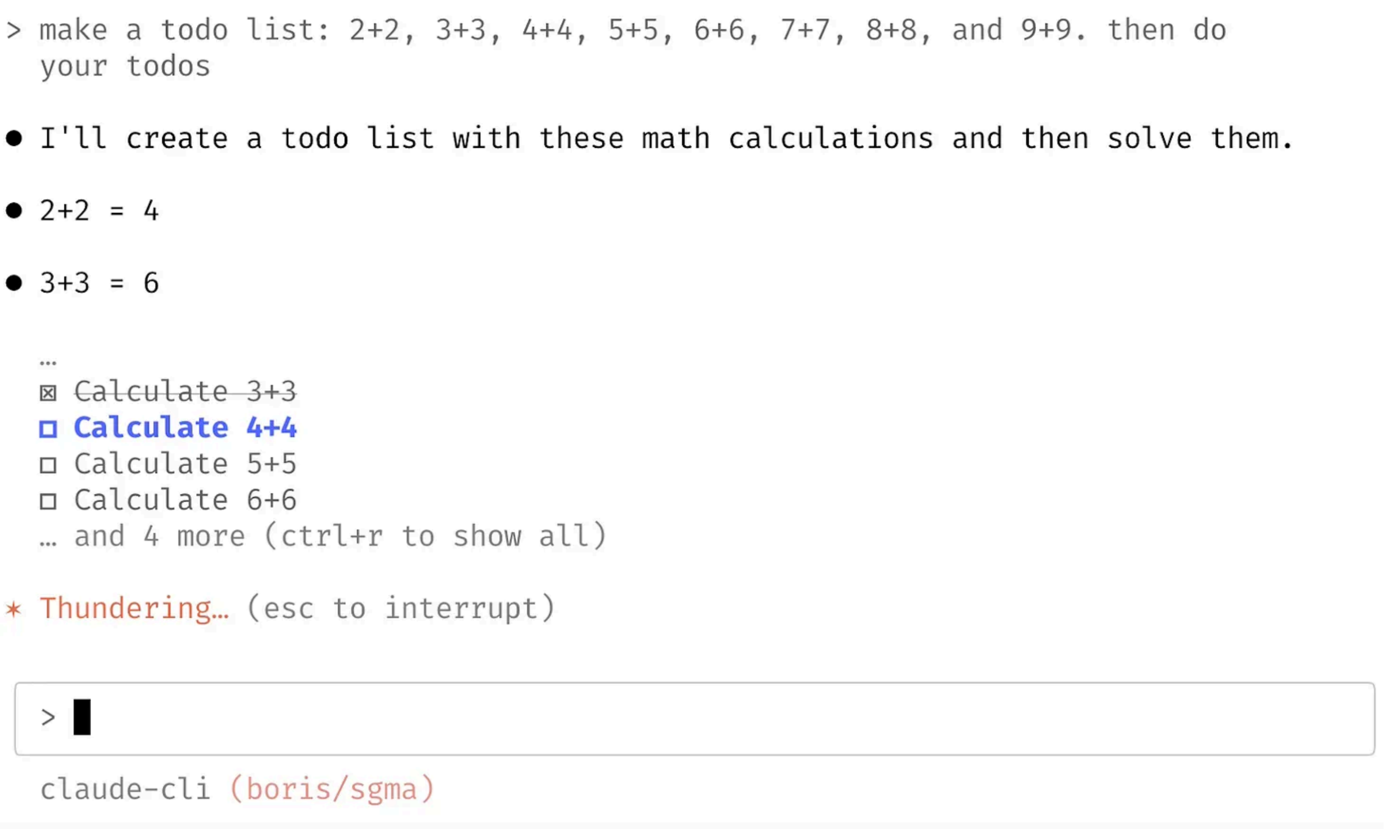
> > truncate at 5 and show "... and 4 more" or whatever



```
● I'll create a todo list with those calculations and then complete them.

● Now I'll complete each calculation:

● 2+2 = 4

● 3+3 = 6

  ☒ ~~Calculate 3+3~~
  ☐ Calculate 4+4
  ☐ Calculate 5+5
  ☐ Calculate 6+6
  …and 4 more · Ctrl+R to show all

✶ Gallivanting… (esc to interrupt)

 > █

  claude-cli (boris/sgma)
```

Prototype #8

> > add a heading "Todo:" in grey text

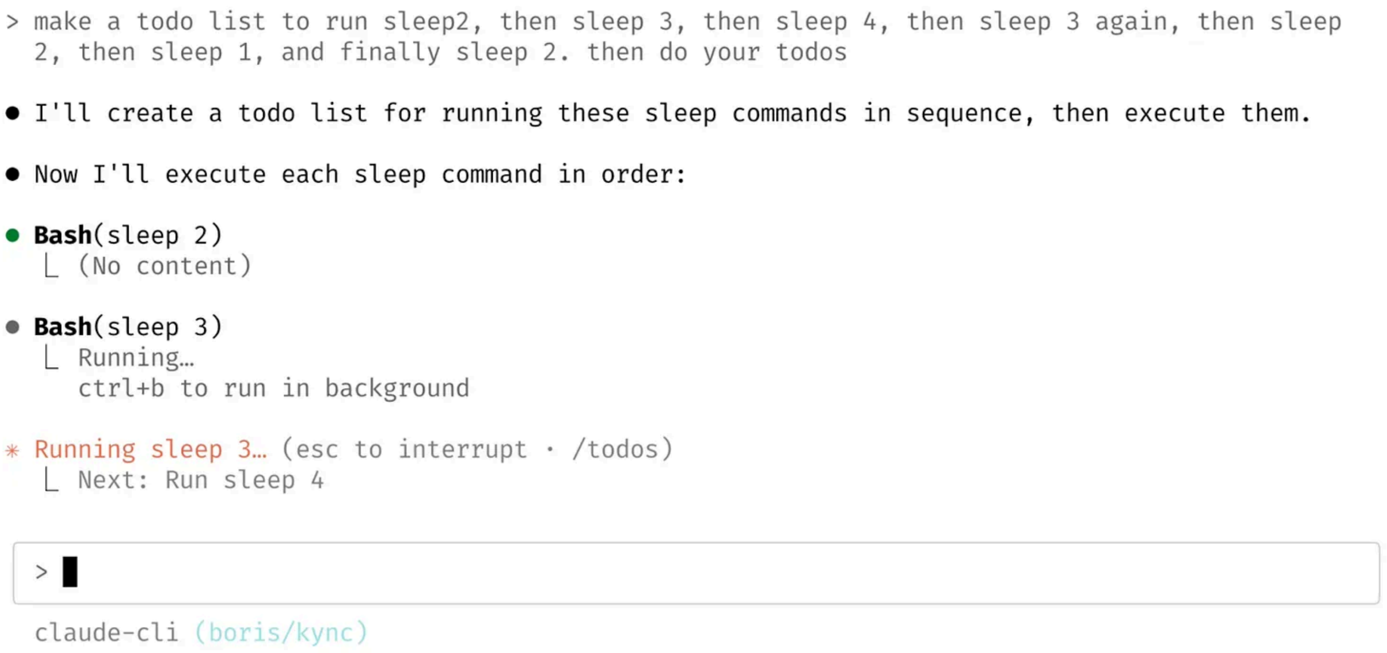


```
> make a todo list: 2+2, 3+3, 4+4, 5+5, 6+6, 7+7, 8+8, and 9+9. then do
  your todos

● I'll create a todo list with these math calculations and then solve them.

● 2+2 = 4

● 3+3 = 6

  …
  ☒ Calculate 3+3
  ☐ Calculate 4+4
  ☐ Calculate 5+5
  ☐ Calculate 6+6
  … and 4 more (ctrl+r to show all)

✶ Thundering… (esc to interrupt)

 > █

  claude-cli (boris/sgma)
```

Prototype #9

Prototypes #10-20: moving the spinner UI element

Boris kept playing with where the todo list visibility should live, and after several more prototypes. In the end, Boris moved the to-do lists to the spinner which maximized visibility, and started to feel good. After a few iterations, they had the version that they ended up shipping in public.

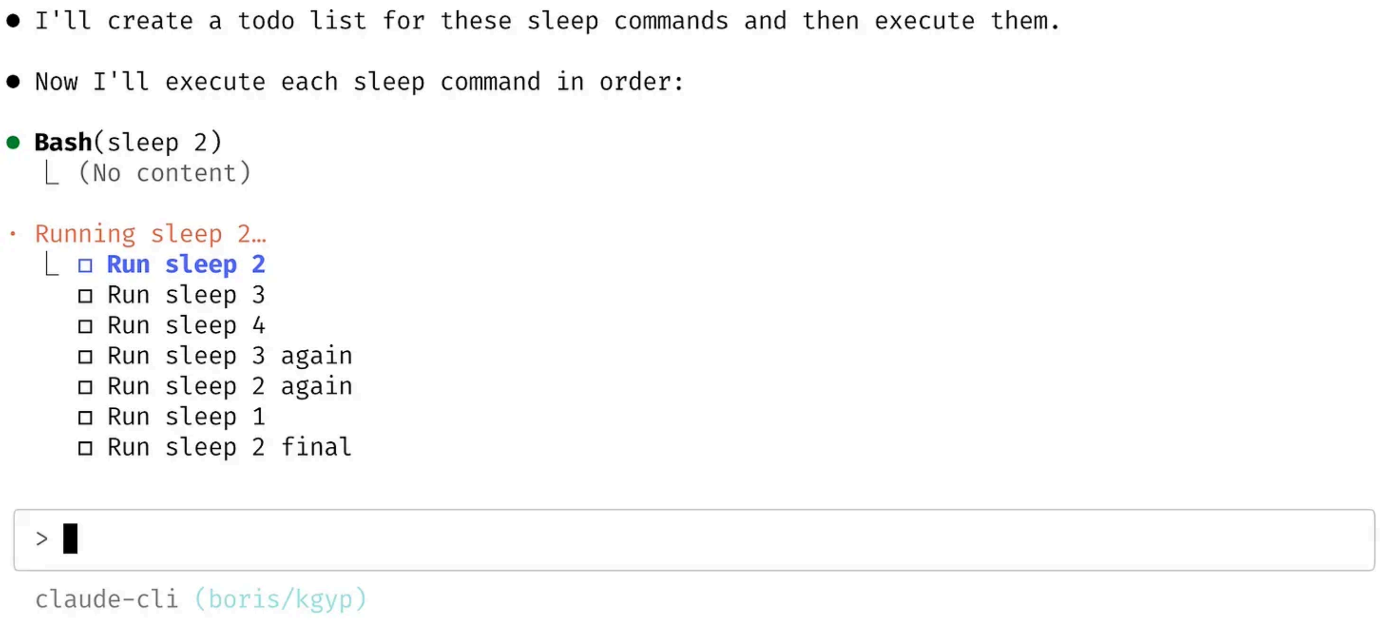


```
> make a todo list to run sleep2, then sleep 3, then sleep 4, then sleep 3 again, then sleep
  2, then sleep 1, and finally sleep 2. then do your todos

● I'll create a todo list for running these sleep commands in sequence, then execute them.

● Now I'll execute each sleep command in order:

● Bash(sleep 2)
  ⎿ (No content)

● Bash(sleep 3)
  ⎿ Running…
     ctrl+b to run in background

✳ Running sleep 3… (esc to interrupt · /todos)
  ⎿ Next: Run sleep 4

 > █

 claude-cli (boris/kync)
```

At around prototype #20, after playing with visibility and the spinner

One more iteration

The team got a lot of feedback from the community that they wanted to be able to see all the todos. So the team added the ability to toggle them with CTRL + T. And this is what’s live today!



```
● I'll create a todo list for these sleep commands and then execute them.

● Now I'll execute each sleep command in order:

● Bash(sleep 2)
  ⎿ (No content)

· Running sleep 2…
  ⎿ ☐ Run sleep 2
     ☐ Run sleep 3
     ☐ Run sleep 4
     ☐ Run sleep 3 again
     ☐ Run sleep 2 again
     ☐ Run sleep 1
     ☐ Run sleep 2 final

 > █

 claude-cli (boris/kgyp)
```

This iteration (around #21 or so) is currently in production – pressing Tab toggles the list of steps executing

Building and testing 5-10 prototype ideas in a day is possible with AI agents. Prototyping used to be so time consuming that if there were two days to prototype, it was lucky to have two distinct prototypes built by the end. But now, agents can build prototypes very quickly, so tests of 5-10 prototypes per day are easily done, as the Claude Code team did.

I don’t suggest everyone can build that many prototypes so quickly, but I think it’s sensible to forget about how long prototyping used to take: these tools change how fast prototyping can be!

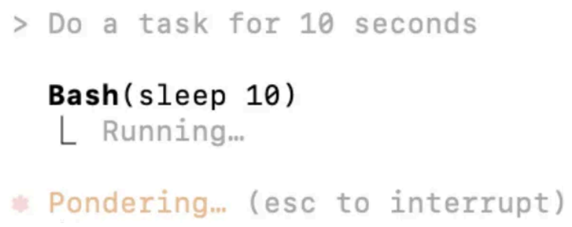
A lot of this prototyping was about making the UI “feel good:” You can see the animated prototypes in this thread. I suggest watching the videos of the prototype steps for a sense of how the feature evolved, and how Boris kept trying new ideas to narrow it down to how the todo list looks inside the tool today.

4. Redesigning the terminal UX

Claude Code has a lot of fresh ideas from the team because this is the first time the terminal is really interactive, thanks to an LLM responding to each command. A few examples:

Contextual loading messages

When the model is thinking, it generates a little loader message based on the work it's doing, such as:



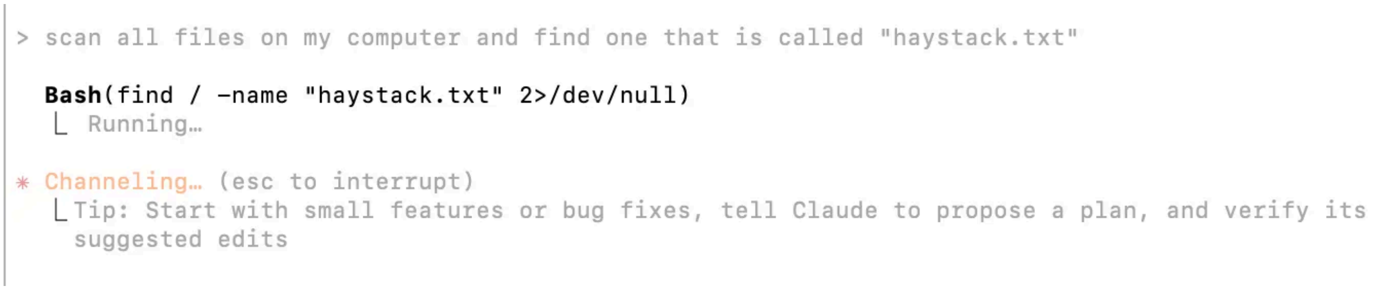
```
> Do a task for 10 seconds

  Bash(sleep 10)
  ⎿ Running…

✻ Pondering… (esc to interrupt)
```

The contextual loading message

This is a small touch, but because the word relates to the action the model is doing, it gives the tool more personality than your average dev tool. To me, this makes it a little more relatable or appealing. *The last time I saw this kind of playful interaction was with Slack's surprisingly fun onboarding flow. At least, it's enjoyable the first time.*

Tooltips

While Claude is doing a long-running task, it shows tooltips to educate the user about how to use it. Command lines working with commands – clever onboarding!



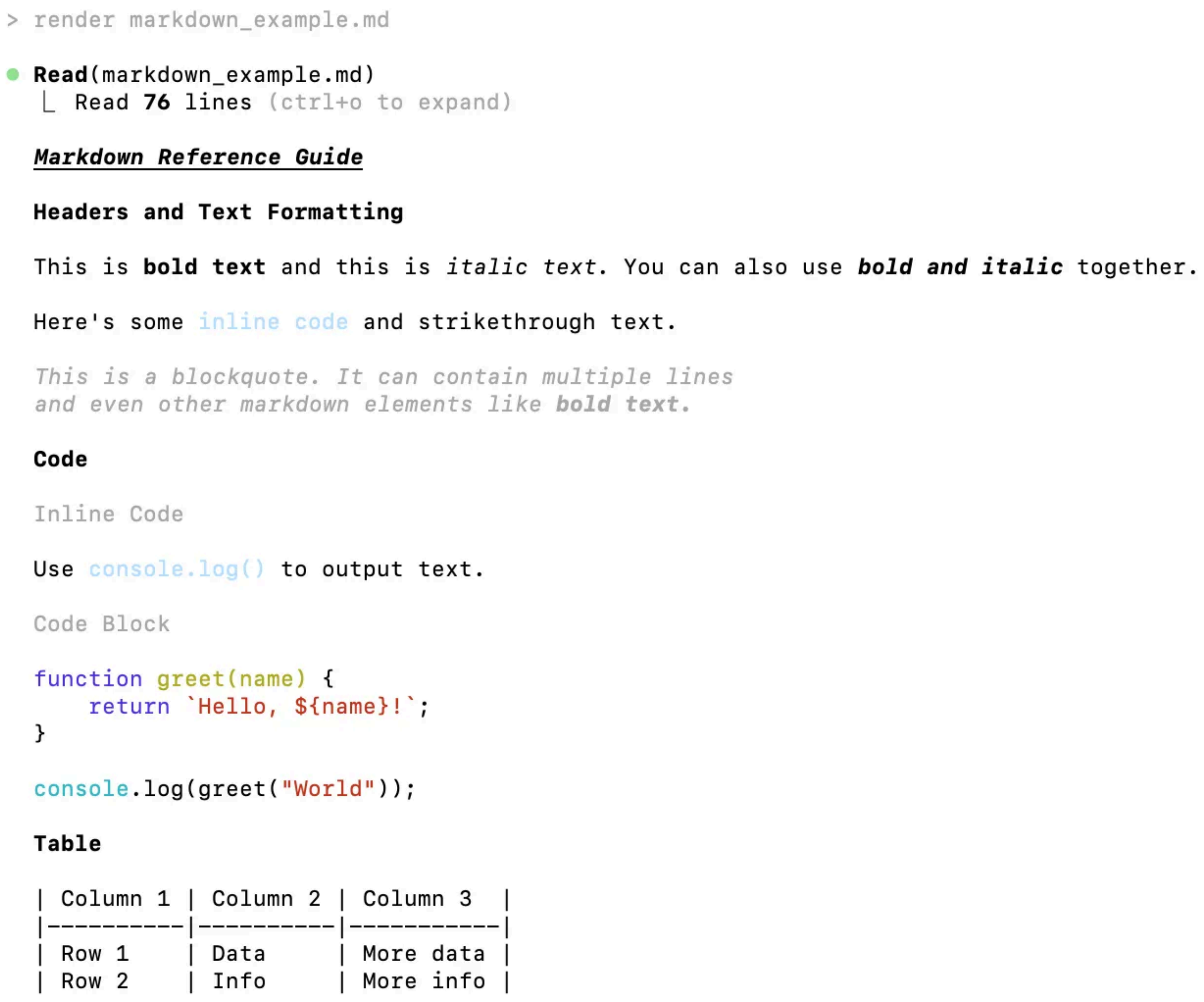
```
> scan all files on my computer and find one that is called "haystack.txt"

  Bash(find / -name "haystack.txt" 2>/dev/null)
   ⎿  Running…

✳ Channeling… (esc to interrupt)
   ⎿ Tip: Start with small features or bug fixes, tell Claude to propose a plan, and verify its
     suggested edits
```

Tooltips are useful when starting out with Claude Code

A vibe-coded custom markdown renderer

Claude generates Markdown which is rendered in the terminal. Before the release, Anthropic engineers complained that nested lists don't really look that good, and that the spacing between paragraphs was off. The problem was with the markdown renderer. Boris tried all the existing ones, but none looked good inside the terminal.

The day before public release, Boris used Claude Code to vibe code a Markdown parser and renderer. And the result was better than anything preceding it, so they shipped it! Boris says he thinks no other terminal has the same Markdown rendering:



```
> render markdown_example.md

● Read(markdown_example.md)
  ⎿ Read 76 lines (ctrl+o to expand)

  Markdown Reference Guide

  Headers and Text Formatting

  This is bold text and this is italic text. You can also use bold and italic together.

  Here's some inline code and strikethrough text.

  This is a blockquote. It can contain multiple lines
  and even other markdown elements like bold text.

  Code

  Inline Code

  Use console.log() to output text.

  Code Block

  function greet(name) {
      return `Hello, ${name}!`;
  }

  console.log(greet("World"));

  Table

  | Column 1 | Column 2 | Column 3  |
  |----------|----------|-----------|
  | Row 1    | Data     | More data |
  | Row 2    | Info     | More info |
```

Markdown rendering in Claude Code. The Markdown parser and renderer was vibe coded

5. What “AI-first” software engineering looks like

The Claude Code team works very differently to other engineering teams by relying so much on Claude as they develop the product. Some interesting areas:

Code reviews, security reviews, tests, and more

Claude is doing a lot of work that used to be for software engineers:

- **The *first* pass for all code reviews** – not just on the Claude Code team, but across Anthropic. The second pass is always done by a fellow engineer – but

Claude finds a lot of issues in the first pass, say engineers there.

- **Security reviews:** Claude does security reviews of PRs. This feature used to be internal, but in August they released it publicly for all.
- **Writing tests**: the test suite for Claude Code is almost entirely written by Claude. As Claude is so good at writing tests, there's no excuse for not writing them! As Boris tells me: "Back in the day, if an engineer put up a PR with no test, I would have hesitated to comment that they should write a test. It feels a bit like a jerk move. But with Claude, it's a simple prompt to write a test, so we always do it."
- **GitHub issue triaging and a stab at implementing it.** Any time a new issue is created in GitHub, Claude Code does a first pass. Engineers will usually prompt Claude: "Can you implement this?" Boris claims it gets it right on the first shot 20-40% of the time. Fixing small issues is a lot faster.

TDD renaissance

Test-Driven Development (TDD) was popular in the early 2000s, then fell out of favor across the industry in the 2010s. The idea of TDD is to write a test *first*, and then write the production code that makes it pass. This forces you to first think about how to test the code. Proponents of TDD swear that writing tests first does improve the quality of code because if you write the production code first, it's common enough for the tests to be an afterthought, or just retrofitted.

The downside of TDD is that it's a lot more time and effort, and can feel like it's slowing you down; at least this was my experience. Boris says that with Claude Code, he does much more TDD:

> "TDD is a thing that I've always wanted to do. But it's never really worked outside of narrow cases. But with Claude Code, I find myself doing it a ton:
>
> - Before I write a feature, I ask the model to write a test. I also tell the model that the test will fail, and don't try to make it pass.

> - Then I instruct it to make the code change that implements what I want it to do, and also makes the test pass, without changing it
>
> This approach works surprisingly well! When the model has a target to iterate against – like a unit test or a mock – it does a really good job.
>
> **Claude 4.0 is the first series of models where the model can write most of the tests**. With 3.5 and 3.7, we could use it for some of the tests, but it needed a lot of handholding".

Automating incident response

Claude is used by some teams for oncall and incident management:

- **Claude triages incidents.** When an engineer declares an incident, they often turn to Claude Code for triage. It pulls in relevant contacts from Slack, relevant logs from Sentry, and other observability data sources.
- **Finding root causes.** Figuring out the cause of an outage takes time, and the team found that in some settings, Claude Code speeds them up. As Boris tells me: "Claude Code is pretty good at finding incident root causes, and can often do it faster than a human. You can also have it auto bisect, revert changes; of course, all after granting it permission to do so.

Software engineers at Anthropic are in the driver's seat, but delegate a lot of the work to Claude Code. What I find interesting is that it is good at working with systems that us software engineers generally have open in different windows, as we take in the information from different systems to figure out what the root cause of an outage.

Framework migrations with AI

I asked about success stories from engineering teams which use Claude Code. One that's repeated is the speeding up of migrations:

> "One company on Claude Code told us they were migrating from Framework A to Framework B. They estimated the work to take two engineering years, so one

> engineer working on it taking two years; or 10 engineers working on it taking about 2.5 months. They used Claude Code, and it took one engineer two weeks.
>
> I normally would be sceptical of such claims, but we keep hearing similar stories from teams at other companies".

I'd add that *code* migrations seem a very good use case for AI coding tools in general, and especially for agentic AI coding tools. I emphasise *code* migrations because database migrations are a lot more risky and involve more work. *We previously did a deepdive on this topic in "Migrations done well".*

Monitoring & alerting

Monitoring key metrics about a product and alerting for anomalies in order to detect outages, is a given, thanks to the broad usage of Claude Code. But at present, doing so is pretty problematic due to the rapid growth of the product, Boris explains:

> "The product is growing so quickly that our charts monitoring usage detect anomalies all the time. We get alerts set off pretty much every day – simply because we're growing so fast!"

Monitoring charts are put together with Claude Code. The product usage data is stored in Google BigQuery. The team created graphs and charts to monitor product usage, almost entirely using Claude Code. Engineers generate things like ASCII charts and smoother visualizations, and pretty much all BigQuery queries are written by Claude Code, Boris tells me.

Feature flags

Only a few feature flags are used at compile time. Claude Code is used by large numbers of external developers for work. So, the last thing the team wants is to ship a change that breaks the tool for a large number of users. How do they avoid this?

As a reminder, the team at Cursor went about this by using feature flagging cautiously. From the deepdive Building Cursor:

> Cursor uses feature flags in the product, but pretty differently from most startups! Their feature flag implementations were inspired by the Pragma preprocessor in C, according to Sualeh. Feature flags spread across their code are typically for internal use and builds only.
>
> By default, Cursor removes all code guarded by feature flags, thanks to maintaining a list of them. Inside the *product-prod-todesktop.json* configuration file these flags are listed.
>
> Cursor CTO Sualeh says they decided on a much more conservative feature flagging approach to avoid accidentally shipping broken features to customers, with a convenient side effect that Cursor doesn't have to worry about the downside of "regular" feature flags of exposing unreleased functionality for smart developers to find.

Claude Code also uses a similar concept of compile-time flags. As Sid Bidasaria tells me:

> "We use compiled flags to exclude parts of the code from external releases. We also have keyword protections to make sure we don't leak code in the final binary that is not ready to be released".

Feature flags slow the team down, so are not used much. Boris is pretty honest about the "why" here:

> "Do we have feature flags? Once in a while, we use them. But mostly, we're building so quickly that there's just no time for it.
>
> Generally, a new feature is either good enough to ship or it's not. And if it's good enough, we ship it internally and let it soak for a day. If it looks all good, then we release it externally."

At Anthropic, what engineers care most about is, in order:

1. Security

2. Reliability
3. Speed of iteration

So if anything creates too much friction to launch things quickly, the team tries to eliminate it. Many places with high-usage teams might opt for more granular releases, but at Anthropic, if a feature is deemed secure and reliable enough, getting it out the door quickly takes priority. The reason is that this is how they get rapid feedback on it. *It goes back to that at its core, Anthropic is a research lab whose goal is to innovate at a good pace.*

More "parallel coding" work with agents?

Sid told me he regularly runs several Claude Code agents, which work on different coding tasks in parallel. At the end of our conversation, he told me that he'd kicked off a few separate agents at the start of our call, and would now check where those agents ended up.

I was wondering if this is unique to engineers at Anthropic – after all, they have unlimited Claude access, and are also heavily incentivized to experiment with AI.

Anthropic told me that 25% of their Max users are "multi-clauding" on a daily basis. The Claude Max plan is a $100-200/month plan that comes with higher usage limits than the base plan. Anthropic found that a quarter of Max users kick off several Claude Code agents in parallel, similar to how Sid does.

Until now, programming has been a pretty "single-threaded" activity: as an engineer, we work on one thing at a time. The only time we'd context-switch to another task was when waiting on something like a code review, or a long deploy. Doing "parallel" programming has simply not been a thing, at all! But now it could be with AI agents.

There's a lot to learn in this area. Does working with parallel agents only *feel* more productive, or *is* it actually so, what type of work are parallel agents helpful for, and what tradeoffs emerge due to less attention being given to each agent? There are many such question marks. But we also have a new tool to experiment with.

6. Building subagents

A recent feature in Claude Code is custom subagents: setting up specific agents with custom context. Claude Code then uses the custom-defined subagents when it makes sense to, or they can be invoked. For example, you could define:

- A frontend subagent: in the subagent file, specifying it to use React 19, Next.js App Router, and outline the specific development methodology to follow
- A backend subagent with a security focus: spelling out a wide range of security areas to focus on, from HTTP security headers, through cross-site request forgery (CSRF) protection, to database security.
- … and many other subagents. Browse a collection in this GitHub repository

You can have subagents work in the background in parallel. Sid Bidasaria is the engineer who built this feature, and explains how the idea came about:

> "A day before a three-day internal hackathon, I saw a Reddit post along the lines of 'I tried to spin up five instances of Claude Code. I gave them personas, and used the filesystem to communicate between the five personas.
>
> My immediate thought was that this is a really cool use case, but someone shouldn't have to jump through so many hoops to get something like it working. We should probably have something out of the box for this in Claude Code".

Sid started the hackathon with the idea to build what became subagents. It took a few iterations to get there:

Day 1: complex agent topologies. With a small team, they sketched out inter-agent topologies: agents talking to each other using message buses and async agent patterns.

Day 2: back to the drawing board. The team realized they had cool ideas, but they seemed too complex. The problem was that users would have a hard time understanding complex agent communication patterns inside a simple UI like Claude

Code. They started from scratch and asked: “what is the purest form in which everyday developers can use this feature?”

They decided to:

- limit themselves by not “inventing” anything new, but using what Claude Code already had: the “/” command and the .md files
- not implement agent-to-agent communication. Instead, have a simple orchestrator with blocking agents

They also talked with Anthropic researchers who were looking into research about multi-agent patterns, and the research showed that it was inconclusive whether complex agent topologies were beneficial. This was more validation to keep things simple.

Day 3: the simplest implementation. By the end of day 3, the team built the simplest implementation they could come up with, which is close enough to how Subagents launched.

The feature looked good enough and was useful; so, after the hackathon and a bit of cleanup and polish, the team launched the feature in their internal build for all Anthropic staff to use.

7. Future of AI-assisted engineering teams?

Obsessive dogfooding is an ingredient of winning. Around 80% of technical folks at Anthropic use Claude Code daily – which adds up to hundreds of “dogfooders.” They use the internal version of the tool and continuously feed back to the team.

Since most staff are software engineers or researchers, they use Claude Code for coding and software development-related tasks, which is also the biggest use case for Claude Code outside of Anthropic.

Building 5+ prototypes per day is now possible, so embrace it! When Boris told me he built and tested 20 fully functional prototypes for TODO lists in Claude Code, I assumed it must've taken a week or two, with 2-3 prototypes per day at most. But it took two days!

With agentic tools, generating a working prototype can be done in 30-60 minutes; basically, a few iterations with the tool. This means engineers who embrace these tools can iterate much faster than previously.

Software teams can ship faster now

The average engineer on the Claude Code team ships around 5 pull requests (PRs) per day. This is a *lot* compared to how most companies work! Back when I was at Uber, we rebuilt the iOS and Android app under lots of pressure. We moved as fast as possible, yet still:

- it was a decent day to get a PR merged – and a good one when it was a large PR
- it was an above-average day to get 2 medium-sized PRs merged
- it was a great day to get anything more done, which was rare even at crunchtime

One thing we did at Uber was stacked diffs: putting together large PRs from several smaller diffs. Still, outside of "crunch projects," it was rare to see engineers regularly shipping more than one PR per day. Back in 2022, Uber internally set a target of around one PR shipped per engineer, per day (5 a week), as we cover in How Uber is measuring engineering productivity.

Mockups feel like a thing of the past when you can build 5-10 working prototypes per *day*. Boris built 20 different versions of the todo list feature, and played around with all of them. No designer was involved, he did no mockups, and went straight for the prototype itself. He could only build so many prototypes because of how productive he was with the AI agents.

AI tools everywhere in the development process

It's interesting that Anthropic has added AI tools to pretty much all parts of its developer process:

- Writing code: Claude Code
- Writing tests: Claude Code
- Reviewing code: first pass with Claude, second with a human
- Security reviews: first pass with Claude
- Triaging inbound issues: first pass with Claude and Claude Code
- Oncall: the oncall sometimes pings Claude Code to do root cause analysis

As an AI lab, Anthropic naturally tries to use its own AI wherever possible. My sense is that it's working well for this team, which is why they keep using it! And they're seeing more success with the newer models in areas like writing unit tests.

AI tools: source of "developer joy"?

Boris mentioned something really interesting about how fun software engineering has become, as an experienced engineer using AI agents:

> "Software engineering is just so fun now. The last time that I had this feeling was when I started out programming: I was programming graphical calculators in middle school and it felt magical at the time. Coding today makes me feel like that again – which I haven't felt in a long time."

Sid mentioned that moving fast and experimenting gives him more energy than before:

> "There are two things that really energize me:
>
> **The velocity of how fast we're shipping.** Honestly, sometimes it feels like we're shipping too fast! But this pace is really energizing because of the freedom we have: on Claude Code, we can ship whatever we want and experiment with things.
>
> **Additionally, this much experimentation is super energizing.** In my previous jobs, we moved fast, but what we built was pretty "cookie cutter"; as in we knew

> what we were doing and it was just a matter of execution.
>
> Right now, models change every three months and we need to completely rethink how we do things. Plus, we need to keep our eye on what's happening across the industry, so every six months or so we need to make sure our product still makes sense.
>
> We need to worry about two time scales: the short-term (models changing) and the longer-term (is our product still relevant?). It's a dynamic I've not worked with before and it's really exciting".

Product-minded engineers + AI tools = opportunity

One thing that strikes me about Claude Code is that no designers are involved, no UX researchers, and no project managers. The project was started by an experienced software engineer with 17 years of experience, who was formerly a Principal Software Engineer at Meta. He's also previously been a startup founder, the #1 engineering hire at another, and has worked on everything from frontend and dev tools, to code quality. Boris evidently has strong experience and an entrepreneurial mindset.

Other engineers on the Claude Code also bring a lot of experience and autonomy, plus open minds to using AI agents for work and experimenting with them. This small team managed to build a developer tool that seems to have found one of the best product-market fits I've yet seen.

Boris and Sid strike me as engineers who are not just product-minded, but also extremely experimental. Of both I would say they:

- Know how to build quality software and are seasoned practitioners
- Have built products from zero-to-one, and know the upsides of being fast and scrappy
- Are entrepreneurial, and put themselves in the shoes of those for whom they build products

- Are entrepreneurial: come up with ideas and build them
- Are open-minded: use new tools to see how they work, and are open to changing how they work, even if this involves drastic change
- Are impatient: they would rather build a prototype than have a meeting about doing so.

More startups may try to hire similarly product-minded engineers because they can get so much done. AI agents in the hands of a product-minded engineer who's focused on building are very powerful because that person usually has the judgment to know when…

- it's worth experimenting with other approaches and tools – like Boris trying widespread Markdown implementations
- the output is good enough to ship to production – like the Markdown implementation shipped last-minute in Claude Code
- it needs some tweaking – like many prototypes of the todo list
- it needs to be thrown out – such as several other todo list prototypes
- it's time to start from scratch – after each failed todo prototype implementation

A different type of startup

It's great to see the Claude Code team having so much success with their agent, but there's an elephant in the room: Anthropic is a very different startup from what most companies will ever be, and some of its characteristics don't apply more widely:

- **It's a research lab,** not a product company**.** At its core, Anthropic researches new and innovative approaches for AI models.
- **The core product is an LLM.** Anthropic's main product is Claude, its LLM. Claude Code is a wrapper around the model. The goal of all of Anthropic's models is to let users "feel the LLM" and have the LLM do most of the work.

- **Deleting code from Claude Code might not be unique to Anthropic.** The Claude Code team deletes production code with new model releases because Claude is their main product. If the LLM can do something, they'd rather delete the code that did that thing before. On one hand, this could be unique to LLM labs, while on the other, I wonder if we'll see similar things at product companies? Once a model can do a task just as well as a program does, will teams delete deterministic code and offload the work to a model that costs expensive tokens to run?
- **Model cost is not a consideration.** At Anthropic, everyone has full, unrestricted access to the most advanced Claude model, Claude Opus. In contrast, at most companies, AI subscriptions are budget items that are under debate, and it's rare for any startup to grant unlimited usage and spending on them.

Takeaways

Thanks very much to Boris, Sid, and Cat on the Claude Code team for sharing all the details about how they built and keep building this innovative and popular tool.

The Claude Code team works very differently to most engineering teams: they are all-in on AI tools and iterate quickly, while also rediscovering things like TDD. It's clear that using AI tools nonstop fills the team with energy.

Of course, Anthropic's main product is a large language model, so in some ways it's natural that engineers there use LLMs for anything and everything – and then some. But my sense is that these experienced engineers who have built software for 10+ years without AI, really *do* get a big boost from AI agents.

It's hard to predict exactly what software engineering will look like in a few years, but we can see one possible future in how the Claude Code team works today. Of course, as always with taking inspiration from others, consider the highly-specific context of the Anthropic team compared to your own, and take inspiration for new ways of working which fit *your own* needs best.

I hope you found this deepdive as interesting and useful as I did!

---

425 Likes · 34 Restacks

Discussion about this post

Comments Restacks

Write a comment...

Andrew 4d

Liked by Gergely Orosz

We are mostly AI-first now and ripping. Things that helped:

-good AGENT.md / CLAUDE.md etc

-using three-step SDD (spec driven development): high level spec->technical spec->step by step execution plan (markdown with [] that the agents checks when completes)

-TDD (pay attention to the remark by Boris in the article above: once in a while, agents still cheat and try to write failing tests, it's a real thing, extra prompting helps), better logging/telemetry - so the agent could self-verify and self-correct better

-resetting the context when needed (eg not trying to use the same agent to execute all steps of a long plan)

-simplifying our own architecture - what's good for humans is good for agents

-so far we still do full human supervision, not just for the results, but for the intermediate artifacts/process - playing with more/better AI verification. it's time consuming (as both Opus and GPT5 are slow), but running things in parallel helps

-we started experimenting with multi-model: use Zencoder to run Claude Code _AND_ Codex on the same task, then use one of them to merge the results. That gets us one step closer to AI doing bit more with a bit less handholding.

Does anyone have a good luck with AI code review? We still get too many false positives, so it's hard to put that on auto-pilot.

LIKE (9) REPLY SHAI

4 replies

Andrew Rich Andrew Rich 4d

Liked by Gergely Orosz

> Model cost is not a consideration. At Anthropic, everyone has full, unrestricted access to the mos
advanced Claude model

This is absolutely evident in the daily use of Claude Code by non-employees — by running at warp factor 10 at all times, burning through those tokens as if there were no quota, and then hitting a hard stop until the "five-hour limit" resets. I'd love to see a requirement that Claude Code developers work on the product under *normal* usage restrictions, rather than the full, unrestricte
access they have in their day-to-day work.

LIKE (9) REPLY SHAI

1 reply

11 more comments...

© 2025 Gergely Orosz · Privacy · Terms · Collection notice
Substack is the home for great culture