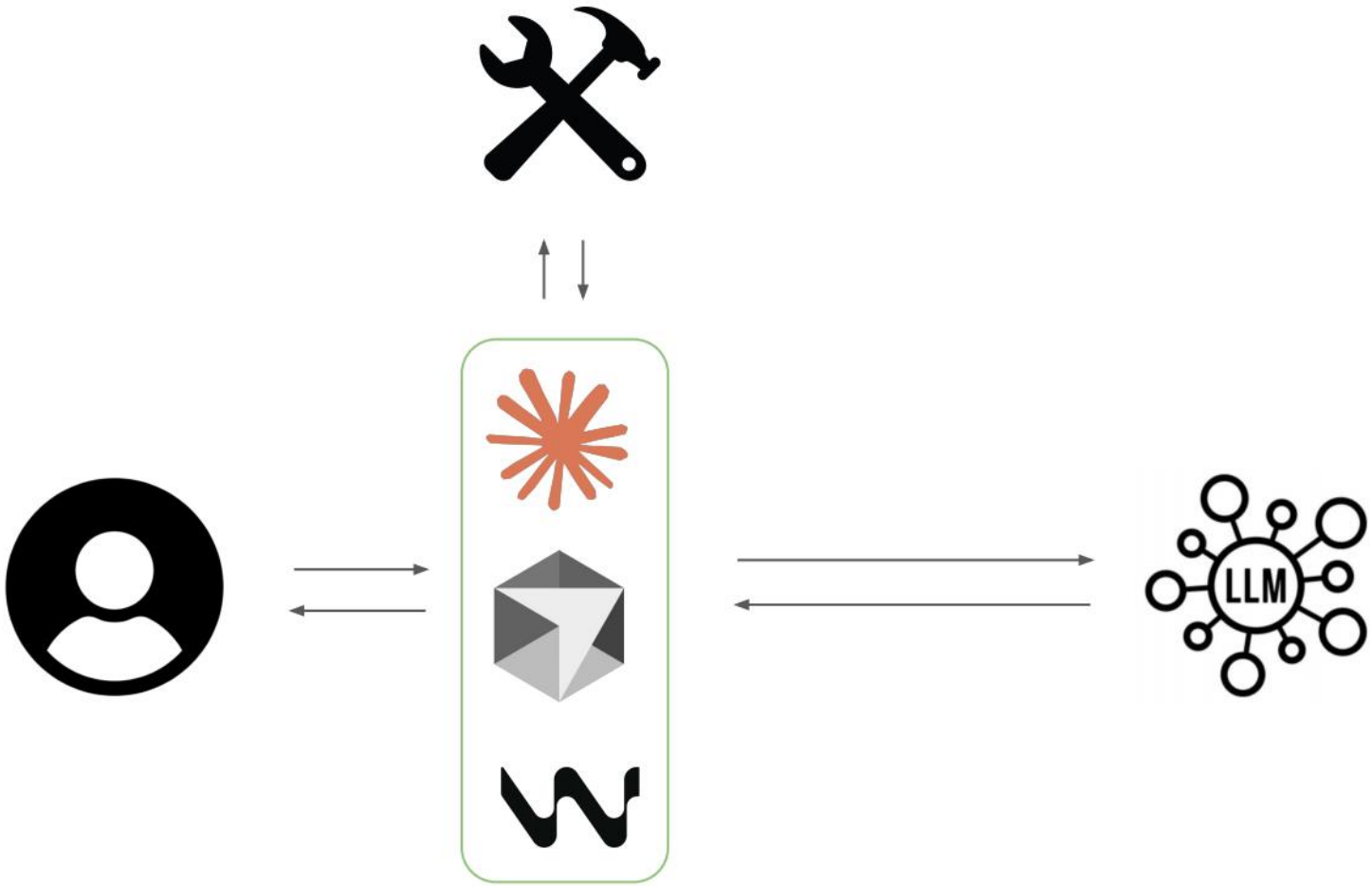
IMEN574/IDSC574 Programing for Data Science

6. **Coding Agents**

Spring 2026

Instructor: **Sangwoo Mo**

It’s that simple



Steps

- Read in terminal and keep appending to conversation
- Tell LLM what tools are available
  - It asks for tool use at appropriate time
  - You execute tool offline and return response
  - “Read_file”
  - “List_dir”
  - “Edit_file”
- Create a new file, edit a new file

The “Secret” Sauce

- Looking under the hood of Claude
  - Front-load context with tiny targeted prompts
  - System reminders everywhere including system/user prompts, tool calls, tool results to prevent drift (<system-reminder> tags)
  - Command prefix extraction
  - Spawns sub agents (likely to help with preventing context overloading)

To MCP and Beyond

Why

- LLMs have vast (but static) world knowledge that only updates when we retrain
- To build fully autonomous systems we need robust ways to feed dynamic data in
  - What's the weather today
  - Who's president
  - What's the price of Bitcoin
  - Who's the narrator in Nike's latest ad campaign
- RAG and tool-calling are the best answer we have today

Basics

- **M**odel **C**ontext **P**rotocol
  - Open protocol that allows systems to provide context to AI models in a manner generalizable across integrations
    - In English: standard format for exposing tools to LLMs
- History: in the distant past pre-November 2024 when MCP was introduced…

Imagine integrating with a questionable 3rd party API



What APIs do you expose?

????

```python
def poorly_documented_twitter_search(bearer_token: str, query: str = "openai"):
    """
    Example function showing how confusing Twitter API v2 felt when it was poorly documented.

    Issues:
    - Parameters like 'query' were ambiguously explained.
    - 'tweet.fields' options were incomplete in the docs.
    - 'max_results' limits were undocumented or inconsistent.
    - Error responses were vague and often unhelpful.
    """

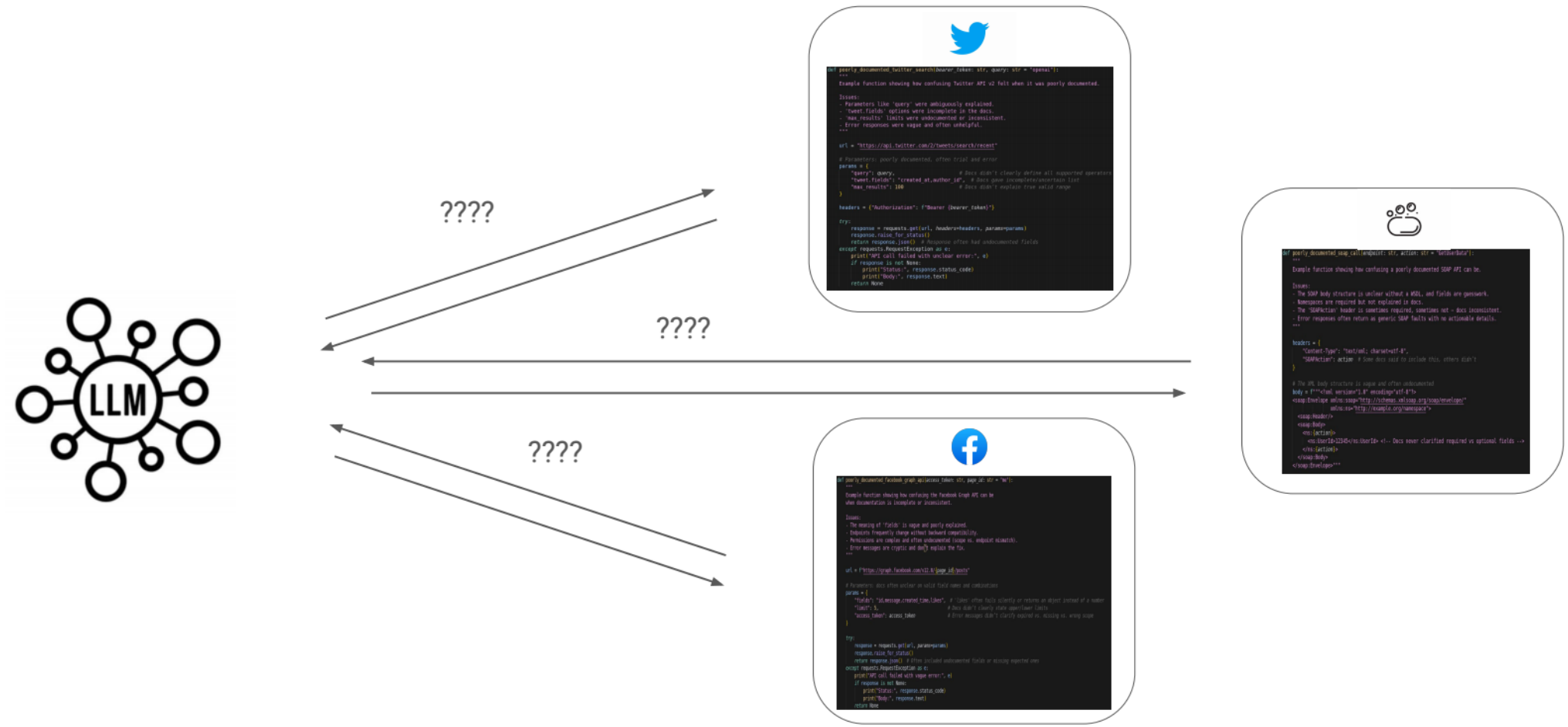
    url = "https://api.twitter.com/2/tweets/search/recent"

    # Parameters: poorly documented, often trial and error
    params = {
        "query": query,                    # Docs didn't clearly define all supported operators
        "tweet.fields": "created_at,author_id",  # Docs gave incomplete/uncertain list
        "max_results": 100                 # Docs didn't explain true valid range
    }

    headers = {"Authorization": f"Bearer {bearer_token}"}

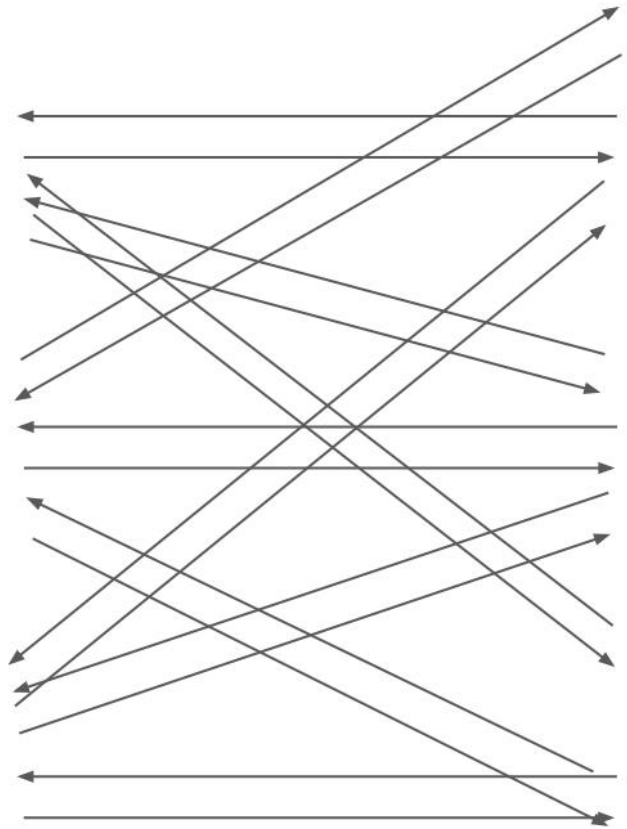
    try:
        response = requests.get(url, headers=headers, params=params)
        response.raise_for_status()
        return response.json()  # Response often had undocumented fields
    except requests.RequestException as e:
        print("API call failed with unclear error:", e)
        if response is not None:
            print("Status:", response.status_code)
            print("Body:", response.text)
        return None
```

Now many APIs



Now many LLM apps











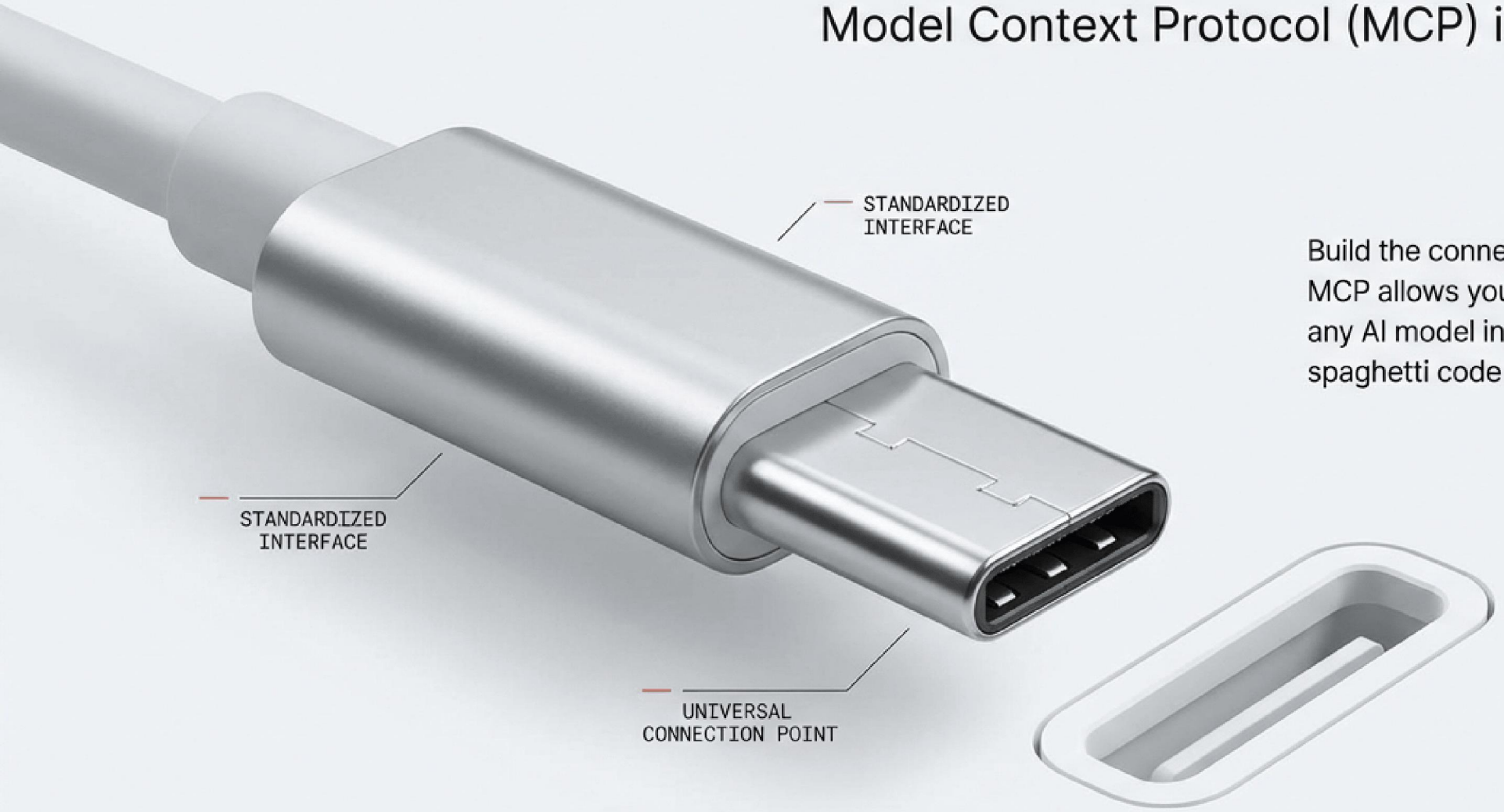
Basics

- MCP
  - Does away with the need to build M x N connectors from LLM host/agent to underlying tool
    - Don’t need to reimplement auth, error handling, rate-limiting, etc
    - Enforces consistent output format using JSON-RPC
  - Extends from Language Server Protocols
    - Allows for proactive agentic workflows rather than purely reactive ones as in LSP
  - Integrating with tools goes from M x N $\longrightarrow$M + N connectors

Swiss
International
Engineering

The Universal Connector

Model Context Protocol (MCP) is the USB-C for AI



Build the connection once. Use it everywhere. MCP allows you to plug any data source into any AI model instantly, replacing custom spaghetti code with a universal standard.

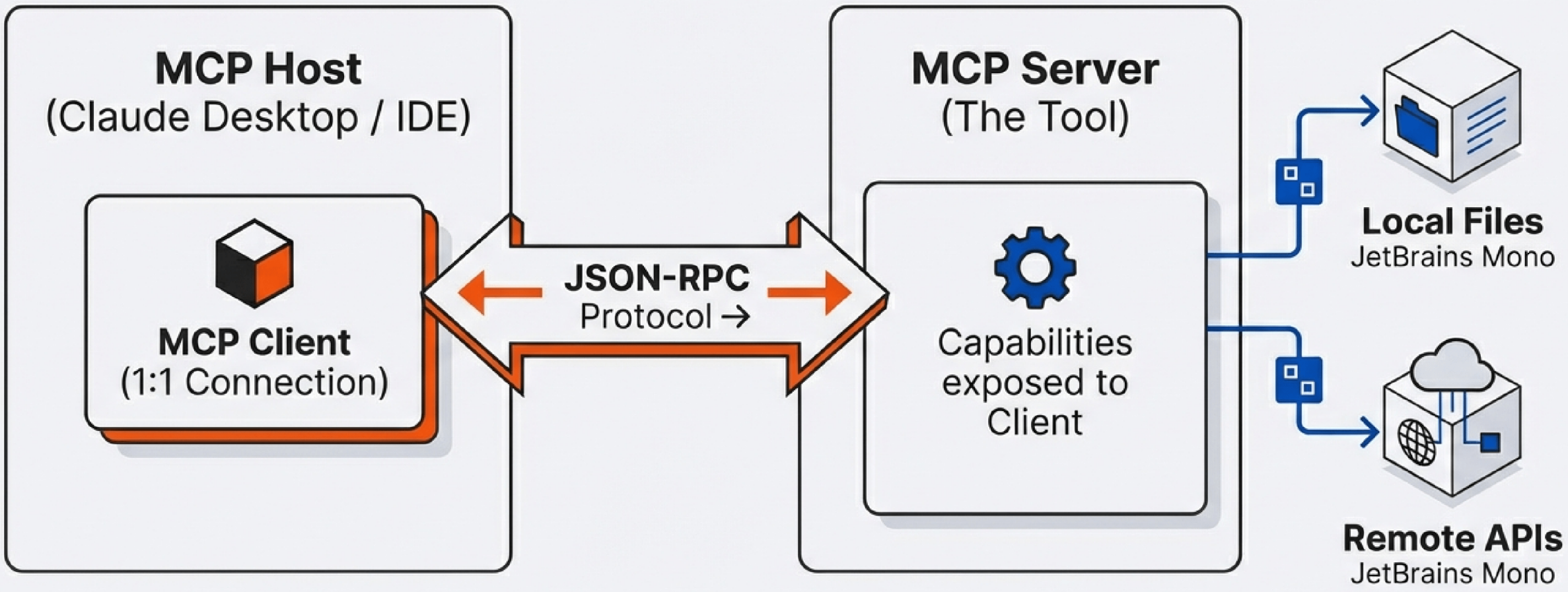
NotebookLM

How It Works: The Restaurant Model



NotebookLM

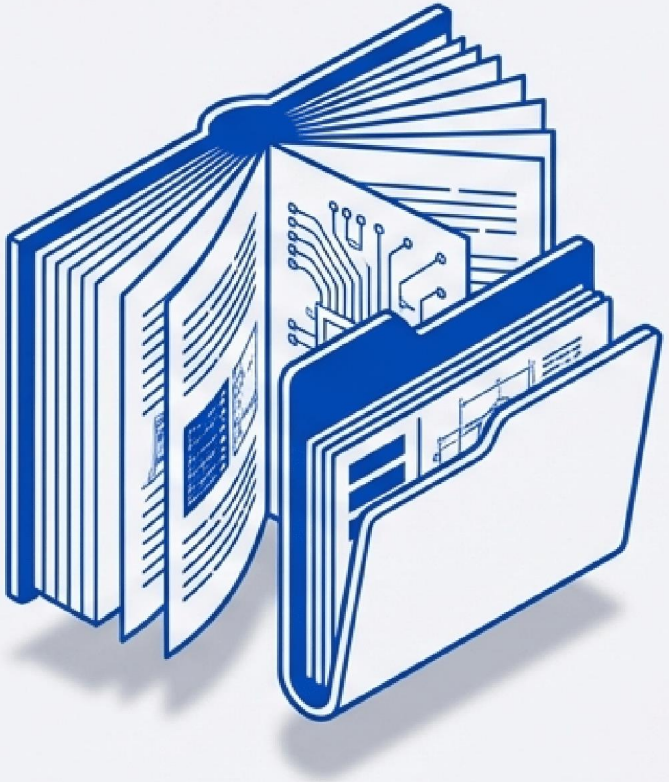
The Client-Server Architecture



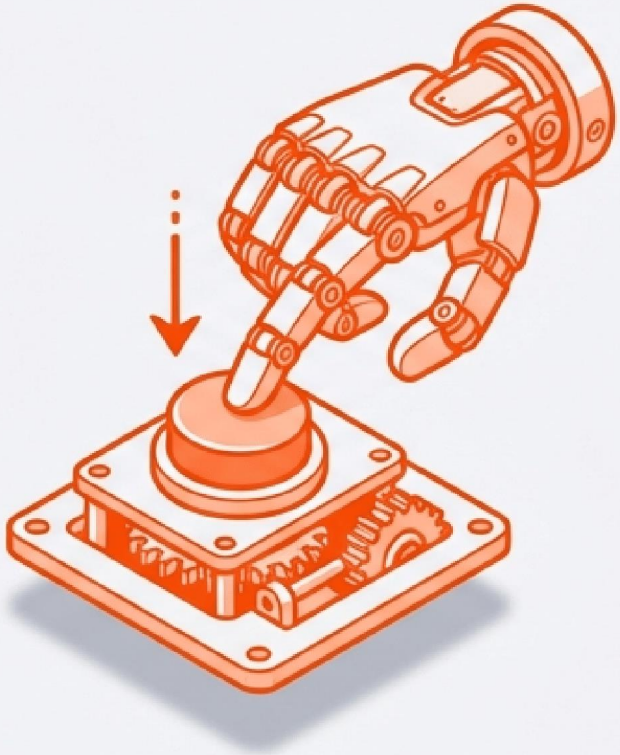
NotebookLM

The Three Building Blocks

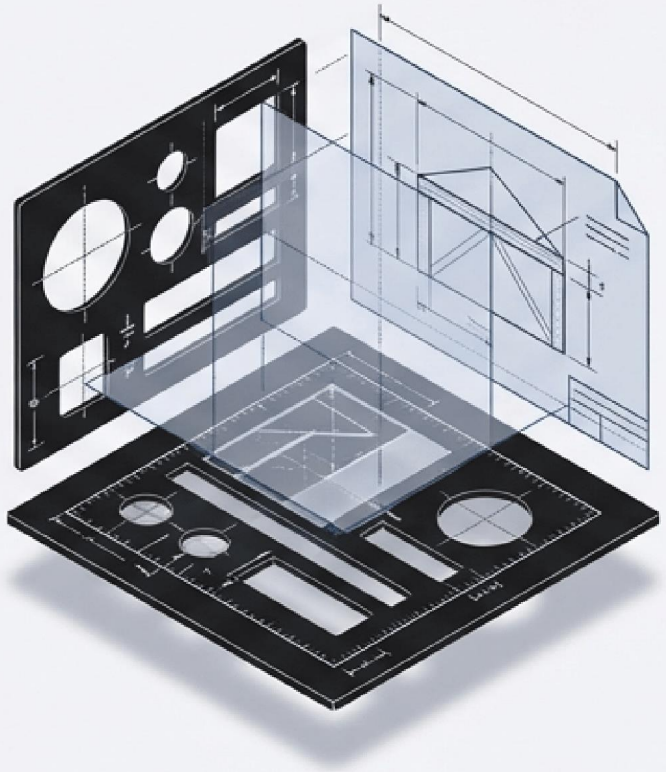
Every MCP Server provides a mix of these three primitives.



Resources



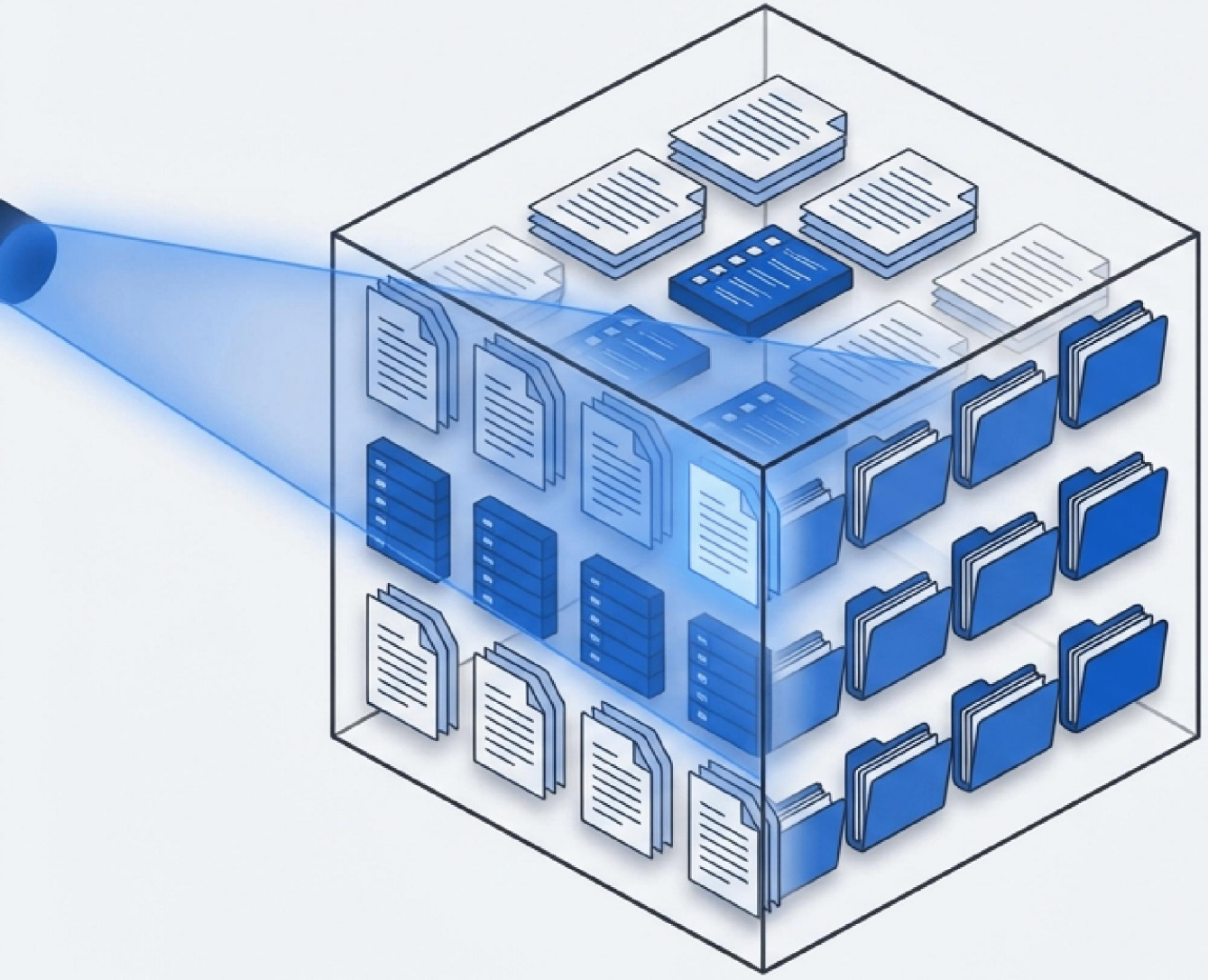
Tools



Prompts

NotebookLM

Resources: Giving the AI ‘Eyes’



Function: Read-Only Data Access.
Use Case: Reading error logs, database rows, or file contents.

```
server.resource('project/logs', read_only=True)
```

NotebookLM

Tools: Giving the AI “Hands”

```
tool: order_pizza(type="pepperoni")
```



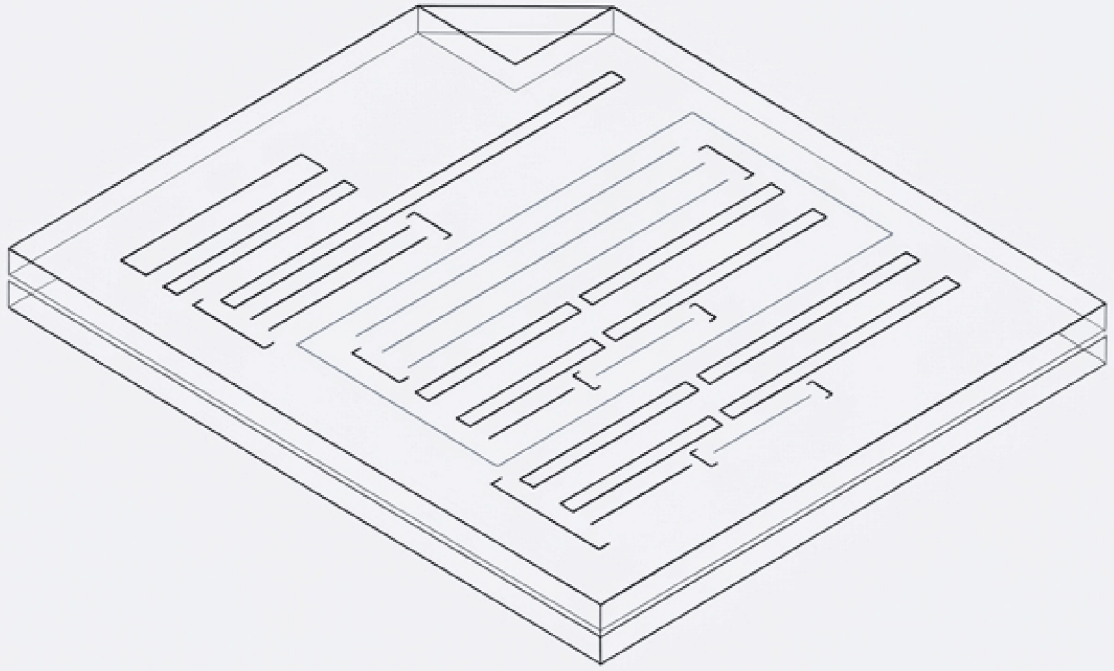
Function: Executable Code.

Use Case: Taking action in the real world.

Example: The user asks for dinner, the AI executes the “Order Pizza” tool via API.

NotebookLM

Prompts: Giving the AI ‘Guidance’



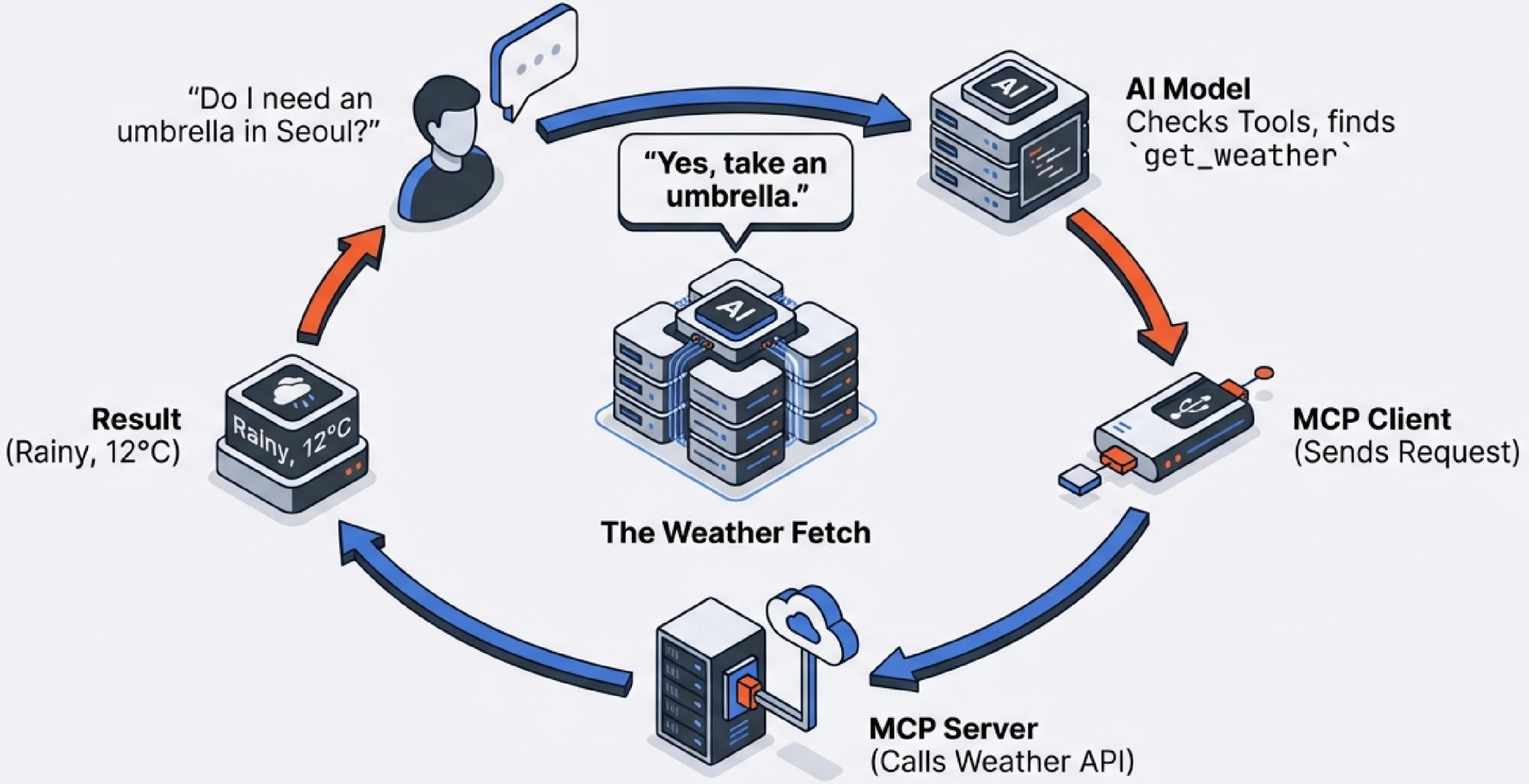
Function: Reusable Templates.

Use Case: Standardizing workflows.

Example: A ‘Bug Report’ prompt that automatically formats error logs into a structure the AI can analyze perfectly.

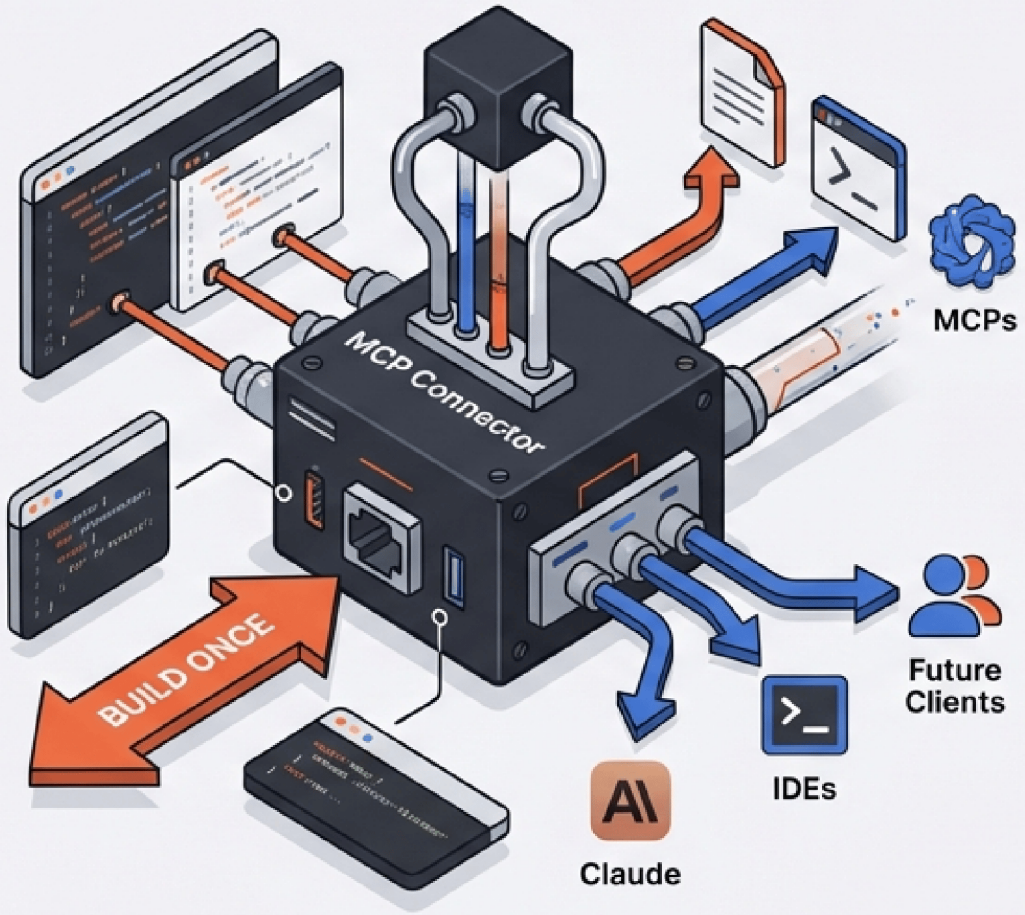
NotebookLM

Scenario: The Real-World Loop



NotebookLM

Why Developers Love MCP



Write Once, Run Everywhere

Build your data connector once. It works with Claude, IDEs, and any future MCP client.

Controlled Security

The AI never sees your root password. You explicitly expose only safe resources.

Model Agnostic

Swap your AI model without breaking your data pipeline.

NotebookLM

Limitations

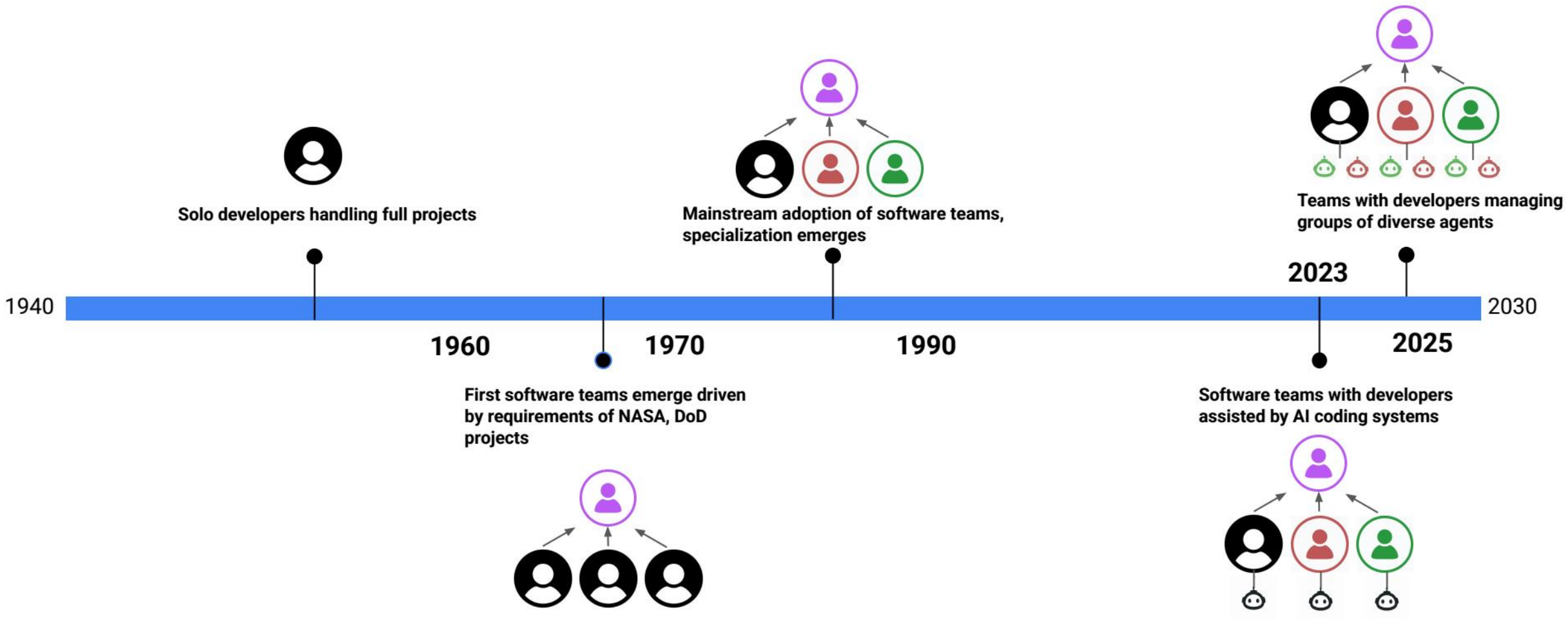
- Agents don’t handle many tools very well today
- APIs eat up your context window quickly
- Design APIs to be AI-native rather that rigid

How to be an Agent Manager

Why

- Development evolution
  - Single developer managing single developer’s output
  - Lead managing many developers worth of output
  - Lead managing many developers worth of output (assisted by an AI system)
  - Single developer managing many AI agents worth of work

A brief history of software teams



Software task steps

- Provide high level requirements 🟩
- Convert requirements into a design doc 🟩/🟦
- Implement solution from doc 🟦
- Add tests 🟦
- Ensure CI (continuous integration) passes 🟦
- Code review 🟦
- Update docs 🟦

Techniques for directing agents

- Agent behavior files (Claude.md/Cursorrules/agents.md)
- Hooks
- Commands
- Subagents

Hooks

- Deterministic scripts that run on predefined event types
  - PreToolUse
  - PostToolUse
  - UserPromptSubmit
  - PreCompact
  - …and more

Commands

- Provide frequently-used prompts as files that the agent can execute
- Use cases
  - Running tests
  - Reviewing code
  - Form a git commit, push

Subagents

- Runtime delegation
- Purposes of a subagent is to
  - Create distinct developer personas for different types of work (frontend, backend, etc)
  - Cleanly separate contexts for different work streams
- Offer
  - Customized system prompts, tools, and a separate context window
  - A move toward agents managing other agents
- Use cases
  - https://github.com/vijaythecoder/awesome-claude-agents/blob/main/CLAUDE.md
  - https://github.com/SuperClaude-Org/SuperClaude_Framework

Best practices

- You need careful backstops
  - Tests in codebase
  - CI/CD best practices
- Auditability of each agent
  - Label every diff made by an agent
- Different models for different classes of tasks
- More complex tasks you may need to handhold a bit more upfront vs fully async ones
- Checkpoint (commit) regularly

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

claude.ai/code

claude code's approach:

works everywhere

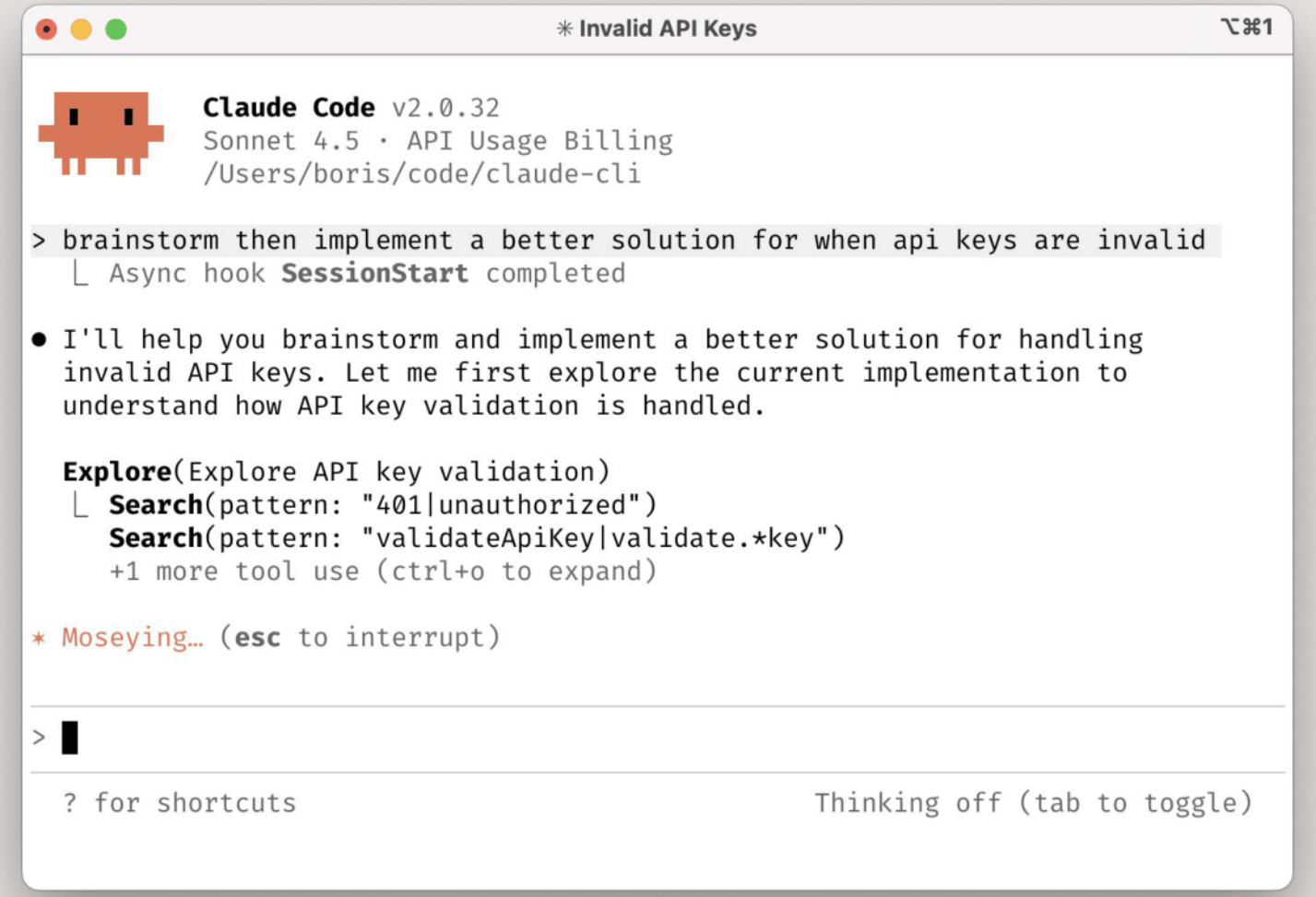
1. terminal-native
2. low-level model access
3. infinitely hackable

works across the whole sdlc

1. Discover	2. Design	3. Build	4. Deploy	5. Support & Scale
Explore codebase and history	Plan project	Implement code	Automate CI/CD	Debug errors
Search documentation	Develop tech specs	Write and execute tests	Configure environments	Large-scale refactor
Onboard & learn	Define architecture	Create commits and PRs	Manage deployments	Monitor usage & performance

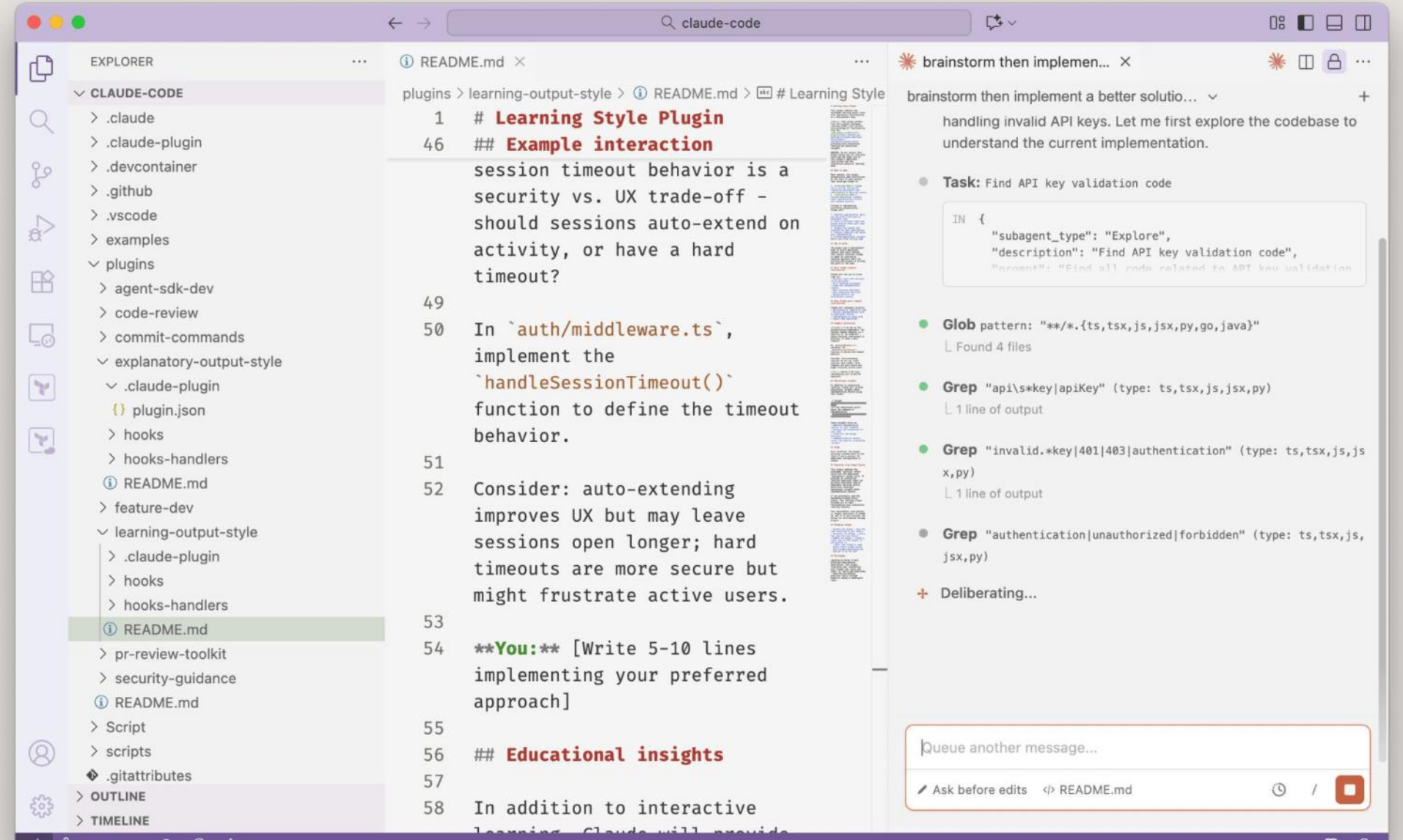
Using and mastering all of your team's CLI tools (e.g., git, docker, bq) so you can focus on solutions, not syntax

one *code, many faces

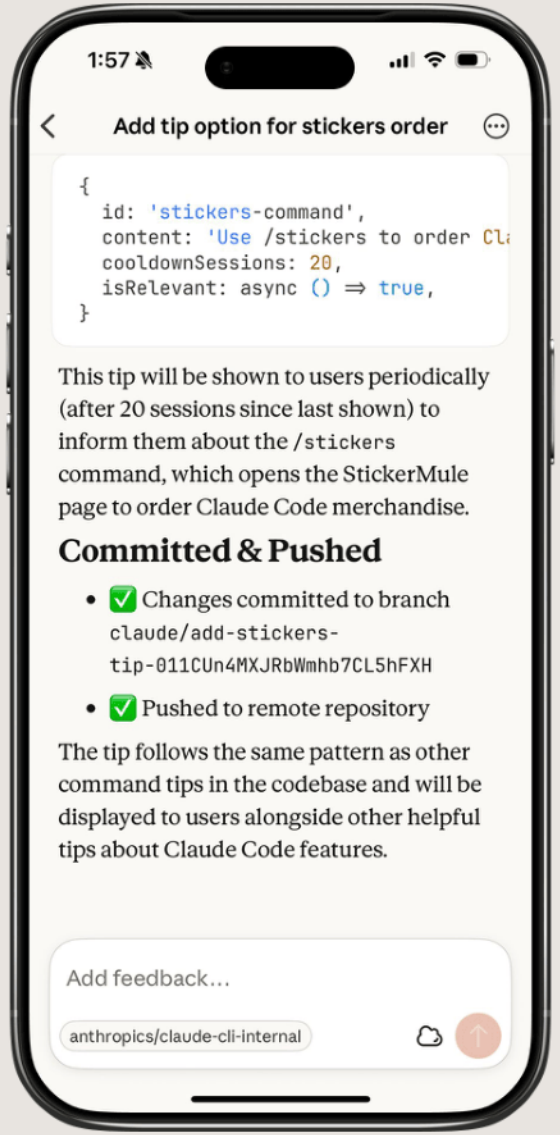

✳ Invalid API Keys
⌥⌘1
Claude Code v2.0.32
Sonnet 4.5 · API Usage Billing
/Users/boris/code/claude-cli
> brainstorm then implement a better solution for when api keys are invalid
└ Async hook SessionStart completed
● I'll help you brainstorm and implement a better solution for handling invalid API keys. Let me first explore the current implementation to understand how API key validation is handled.
Explore(Explore API key validation)
└ Search(pattern: "401|unauthorized")
Search(pattern: "validateApiKey|validate.*key")
+1 more tool use (ctrl+o to expand)
✶ Moseying… (esc to interrupt)
>
? for shortcuts
Thinking off (tab to toggle)

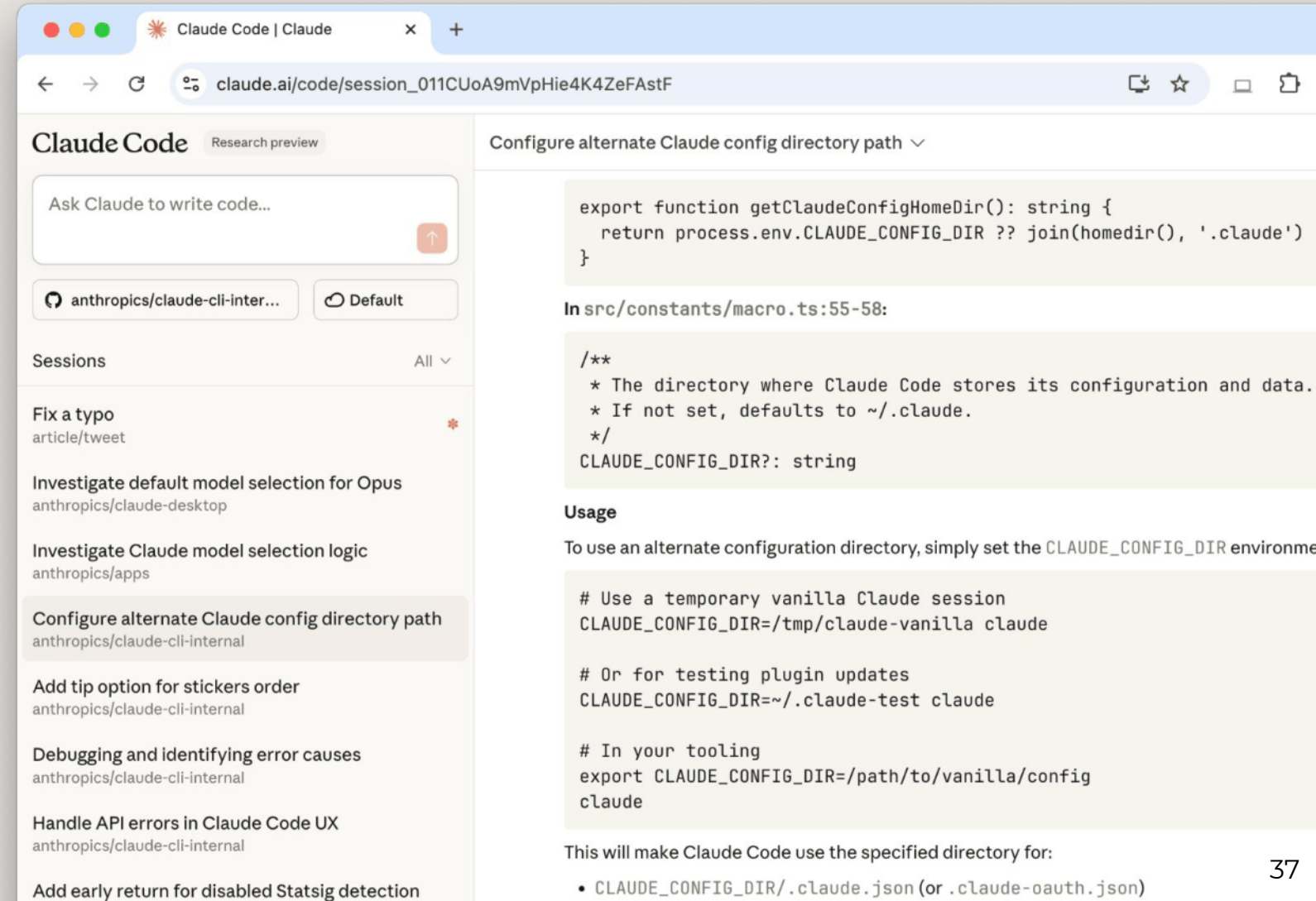
terminal

ide

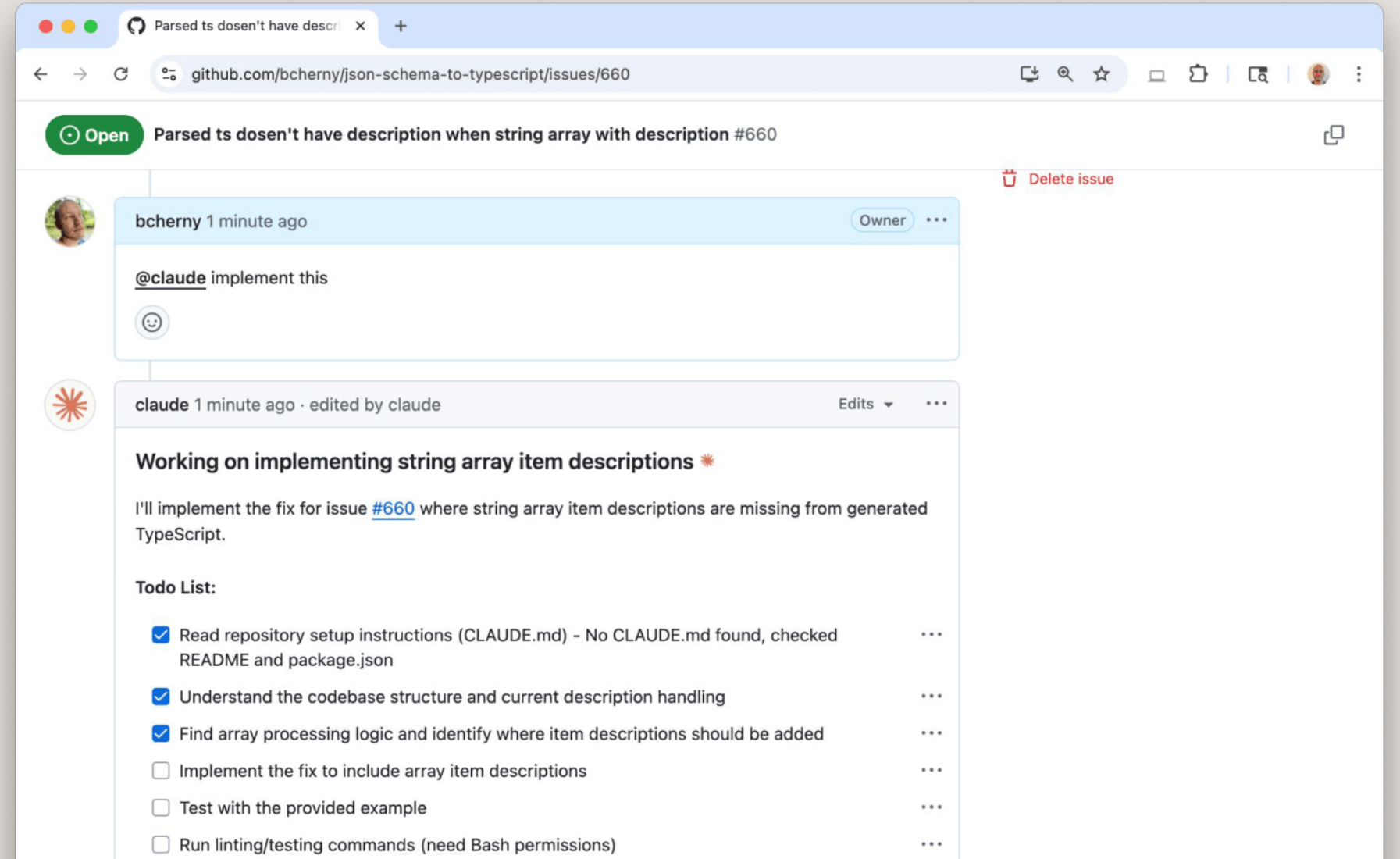

claude-code
EXPLORER
README.md
brainstorm then implemen...
CLAUDE-CODE
.claude
.claude-plugin
.devcontainer
.github
.vscode
examples
plugins
agent-sdk-dev
code-review
commit-commands
explanatory-output-style
.claude-plugin
plugin.json
hooks
hooks-handlers
README.md
feature-dev
learning-output-style
.claude-plugin
hooks
hooks-handlers
README.md
pr-review-toolkit
security-guidance
README.md
Script
scripts
.gitattributes
OUTLINE
TIMELINE
plugins > learning-output-style > README.md > # Learning Style
1 # Learning Style Plugin
46 ## Example interaction
session timeout behavior is a security vs. UX trade-off - should sessions auto-extend on activity, or have a hard timeout?
49
50 In `auth/middleware.ts`, implement the `handleSessionTimeout()` function to define the timeout behavior.
51
52 Consider: auto-extending improves UX but may leave sessions open longer; hard timeouts are more secure but might frustrate active users.
53
54 **You:** [Write 5-10 lines implementing your preferred approach]
55
56 ## Educational insights
57
58 In addition to interactive
brainstorm then implement a better solutio...
handling invalid API keys. Let me first explore the codebase to understand the current implementation.
Task: Find API key validation code
IN {
"subagent_type": "Explore",
"description": "Find API key validation code",
Glob pattern: "**/*.{ts,tsx,js,jsx,py,go,java}"
Found 4 files
Grep "api\s*key|apiKey" (type: ts,tsx,js,jsx,py)
1 line of output
Grep "invalid.*key|401|403|authentication" (type: ts,tsx,js,jsx,py)
1 line of output
Grep "authentication|unauthorized|forbidden" (type: ts,tsx,js,jsx,py)
Deliberating...
Queue another message...
Ask before edits
README.md

web & ios


1:57
Add tip option for stickers order
{
id: 'stickers-command',
content: 'Use /stickers to order Cl
cooldownSessions: 20,
isRelevant: async () => true,
}
This tip will be shown to users periodically (after 20 sessions since last shown) to inform them about the /stickers command, which opens the StickerMule page to order Claude Code merchandise.
Committed & Pushed
✅ Changes committed to branch claude/add-stickers-tip-011CUn4MXJRbWmhb7CL5hFXH
✅ Pushed to remote repository
The tip follows the same pattern as other command tips in the codebase and will be displayed to users alongside other helpful tips about Claude Code features.
Add feedback...
anthropics/claude-cli-internal

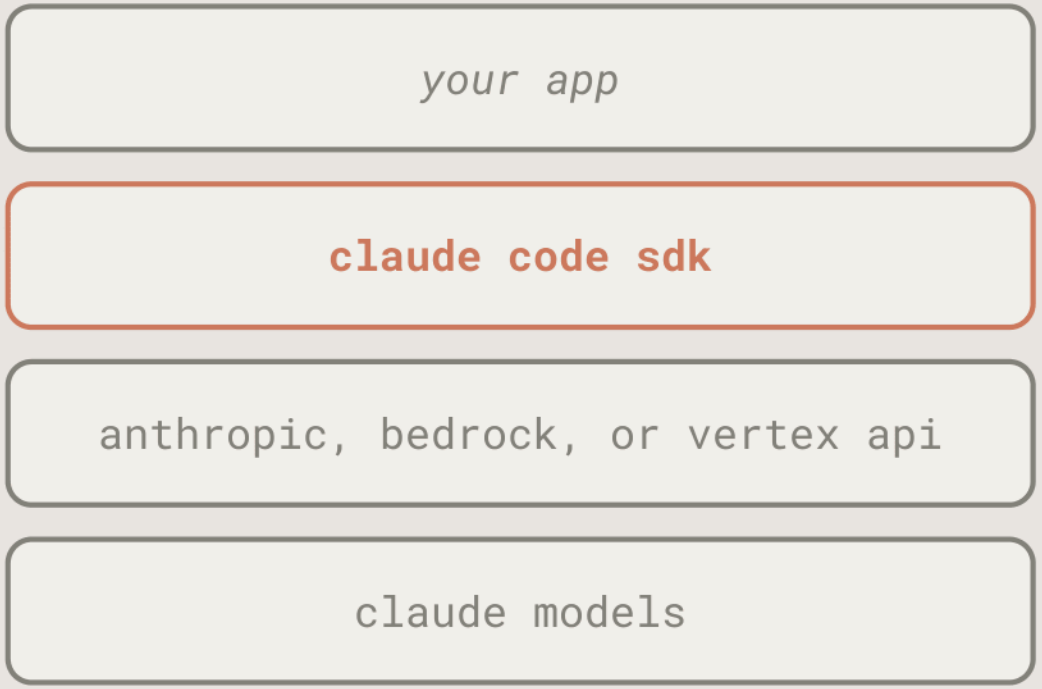

Claude Code | Claude
claude.ai/code/session_011CUoA9mVpHie4K4ZeFAstF
Claude Code
Research preview
Ask Claude to write code...
anthropics/claude-cli-inter...
Default
Sessions
All
Fix a typo
article/tweet
Investigate default model selection for Opus
anthropics/claude-desktop
Investigate Claude model selection logic
anthropics/apps
Configure alternate Claude config directory path
anthropics/claude-cli-internal
Add tip option for stickers order
anthropics/claude-cli-internal
Debugging and identifying error causes
anthropics/claude-cli-internal
Handle API errors in Claude Code UX
anthropics/claude-cli-internal
Add early return for disabled Statsig detection
Configure alternate Claude config directory path
export function getClaudeConfigHomeDir(): string {
return process.env.CLAUDE_CONFIG_DIR ?? join(homedir(), '.claude')
}
In src/constants/macro.ts:55-58:
/**
* The directory where Claude Code stores its configuration and data.
* If not set, defaults to ~/.claude.
*/
CLAUDE_CONFIG_DIR?: string
Usage
To use an alternate configuration directory, simply set the CLAUDE_CONFIG_DIR environme
# Use a temporary vanilla Claude session
CLAUDE_CONFIG_DIR=/tmp/claude-vanilla claude
# Or for testing plugin updates
CLAUDE_CONFIG_DIR=~/.claude-test claude
# In your tooling
export CLAUDE_CONFIG_DIR=/path/to/vanilla/config
claude
This will make Claude Code use the specified directory for:
CLAUDE_CONFIG_DIR/.claude.json (or .claude-oauth.json)

/install-github-app



sdk

```
$ claude -p \
  “what did i do this week?” \
  --allowedTools Bash(git log:*)
  --output-format stream-json
```



using claude code

use cases

1. codebase q&a + research
2. write code
   a. 1-shot
   b. sidekick
   c. prototype
3. integrate tools & mcps
4. power automation

1. ask claude code about your code

> how do I make a new `@app/services/ValidationTemplateFactory`?

> why does recoverFromException take so many arguments? look through git history to answer

> why did we fix issue #18363 by adding the if/else in `@src/login.ts` api?

> in which version did we release the new `@api/ext/PreHooks.php` api?

> look at PR #9383, then carefully verify which app versions were impacted

> what did I ship last week?

2. teach claude to use your tools

```
> use the barley cli to check for error logs
```

```
$ claude mcp add barley_server -- node myserver

> use the barley mcp server to check for error logs
```

3. fit the workflow to the task

explore › plan › confirm › code › commit

> figure out the root cause for issue #983, then propose a few fixes. Let me choose an approach before you code. ultrathink

3. fit the workflow to the task

tests › commit › code › iterate › commit

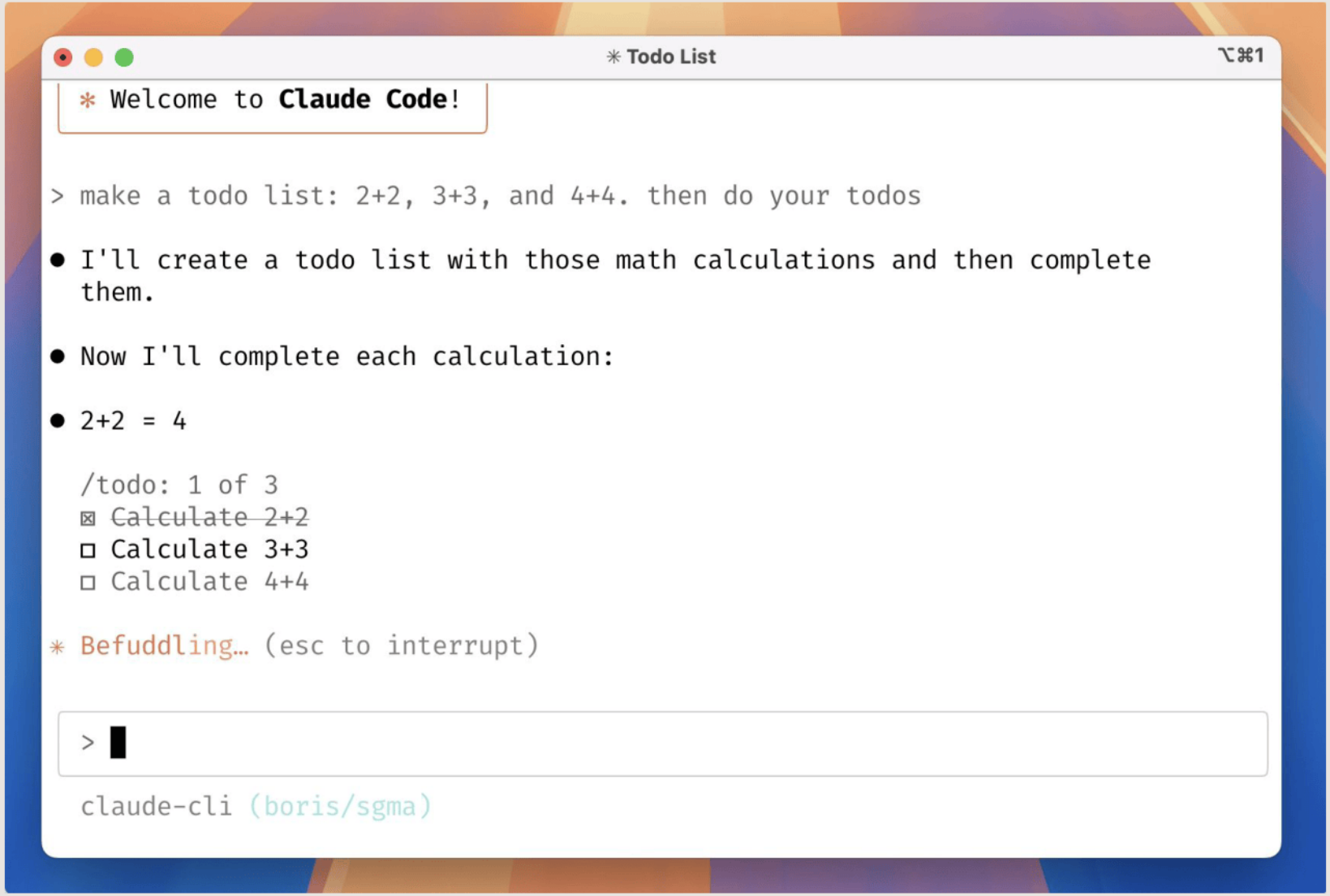
> write tests for `@utils/markdown.ts` to make sure links render properly (note the tests won't pass yet, since links aren't yet implemented). then commit. then update the code to make the tests pass.

3. fit the workflow to the task

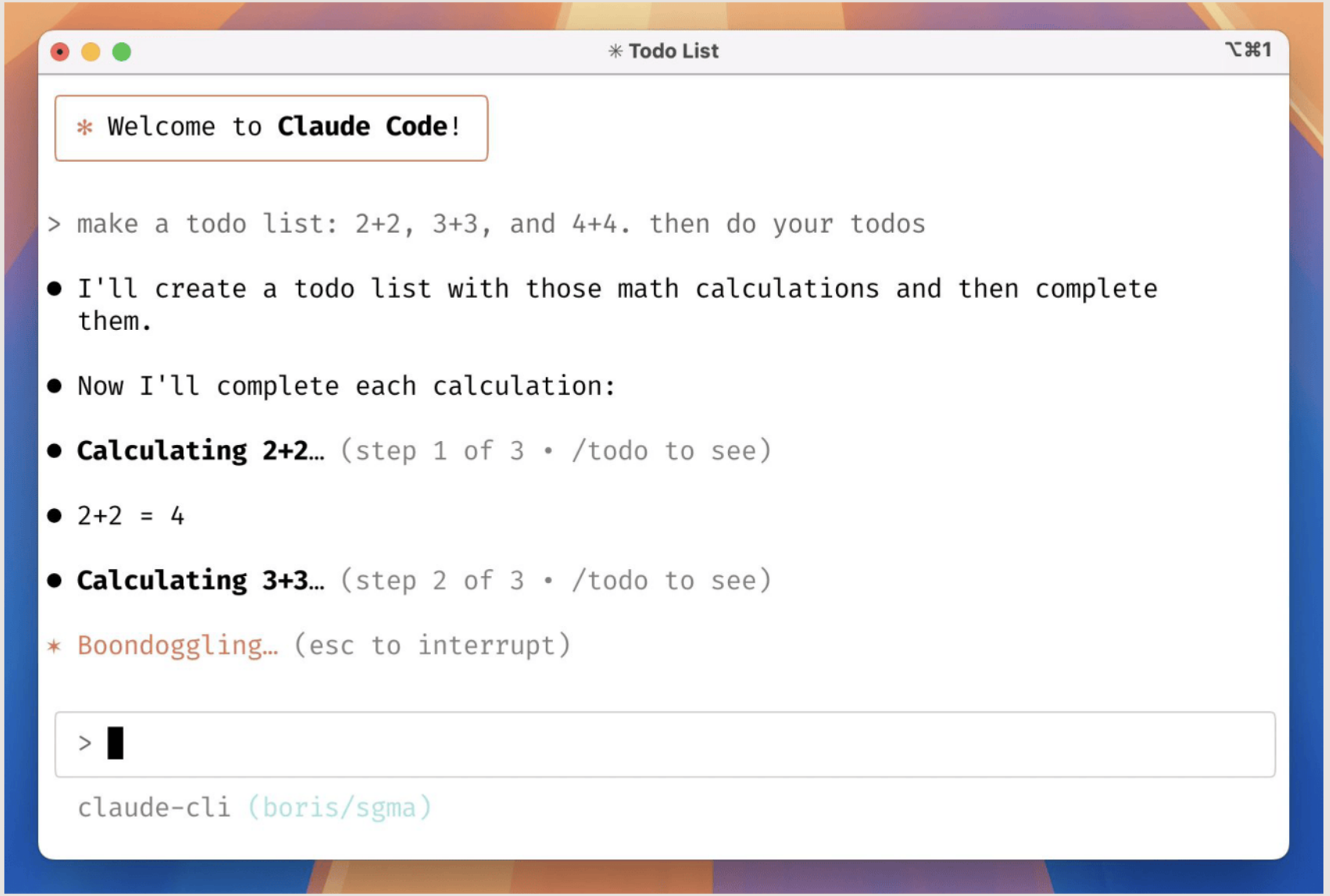
code › screenshot › iterate

> implement [mock.png]. Then screenshot it with puppeteer and iterate till it looks like the mock.

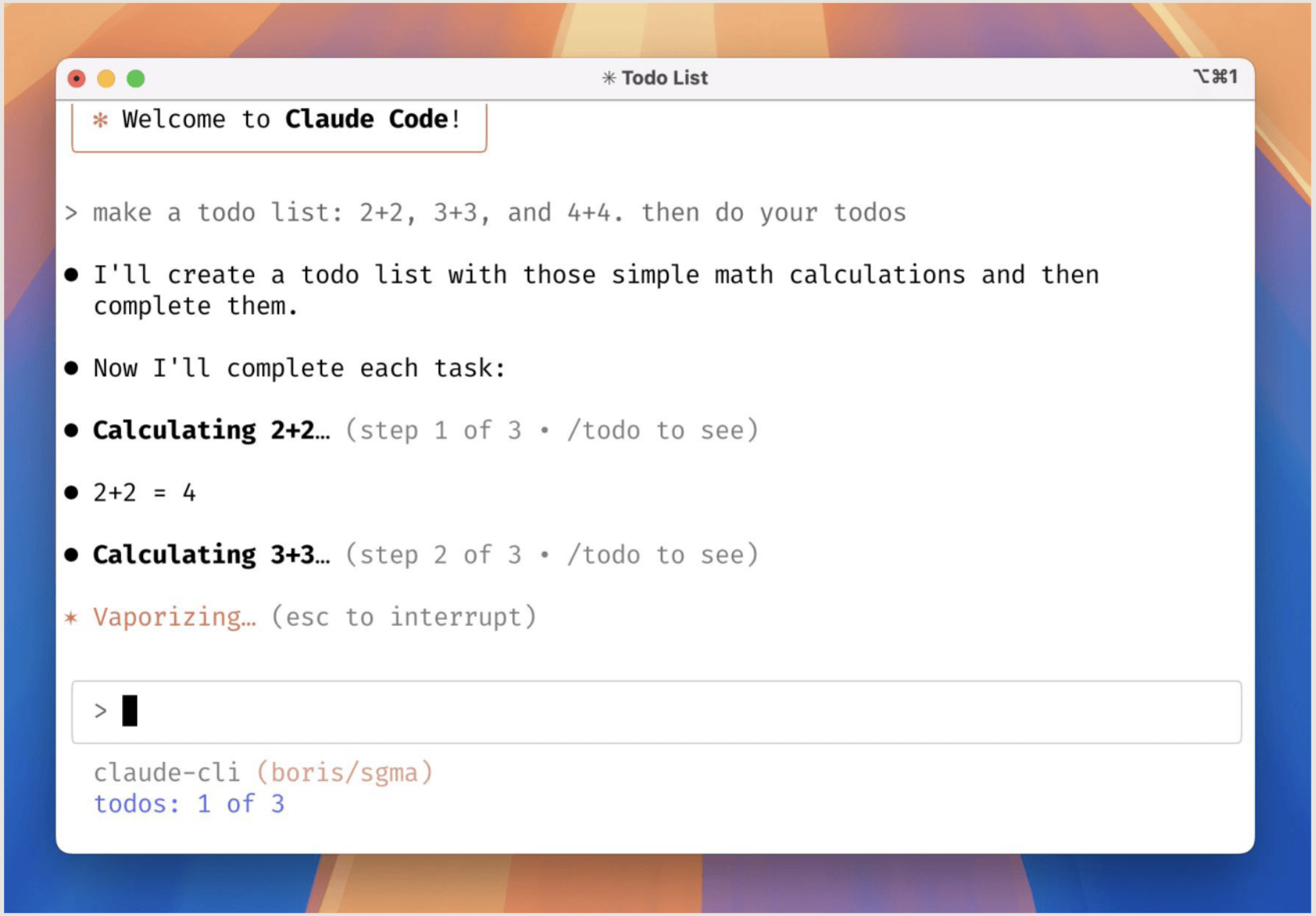
4. use claude code to prototype



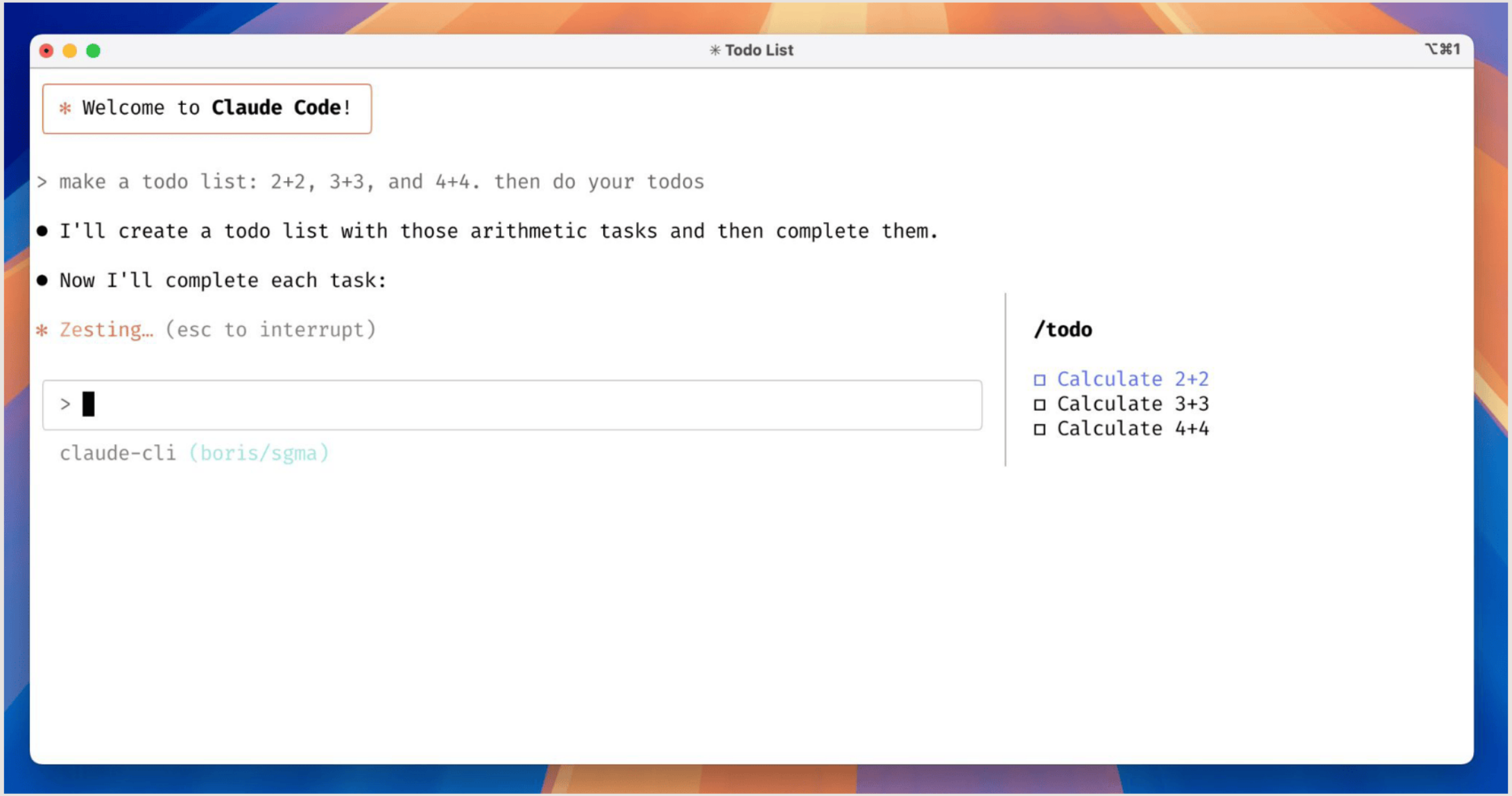
> make it so instead of todos showing up as they come in, we hide the tool use and result for todos, and render a fixed todo list above the input. title it "/todo (1 of 3)" in grey



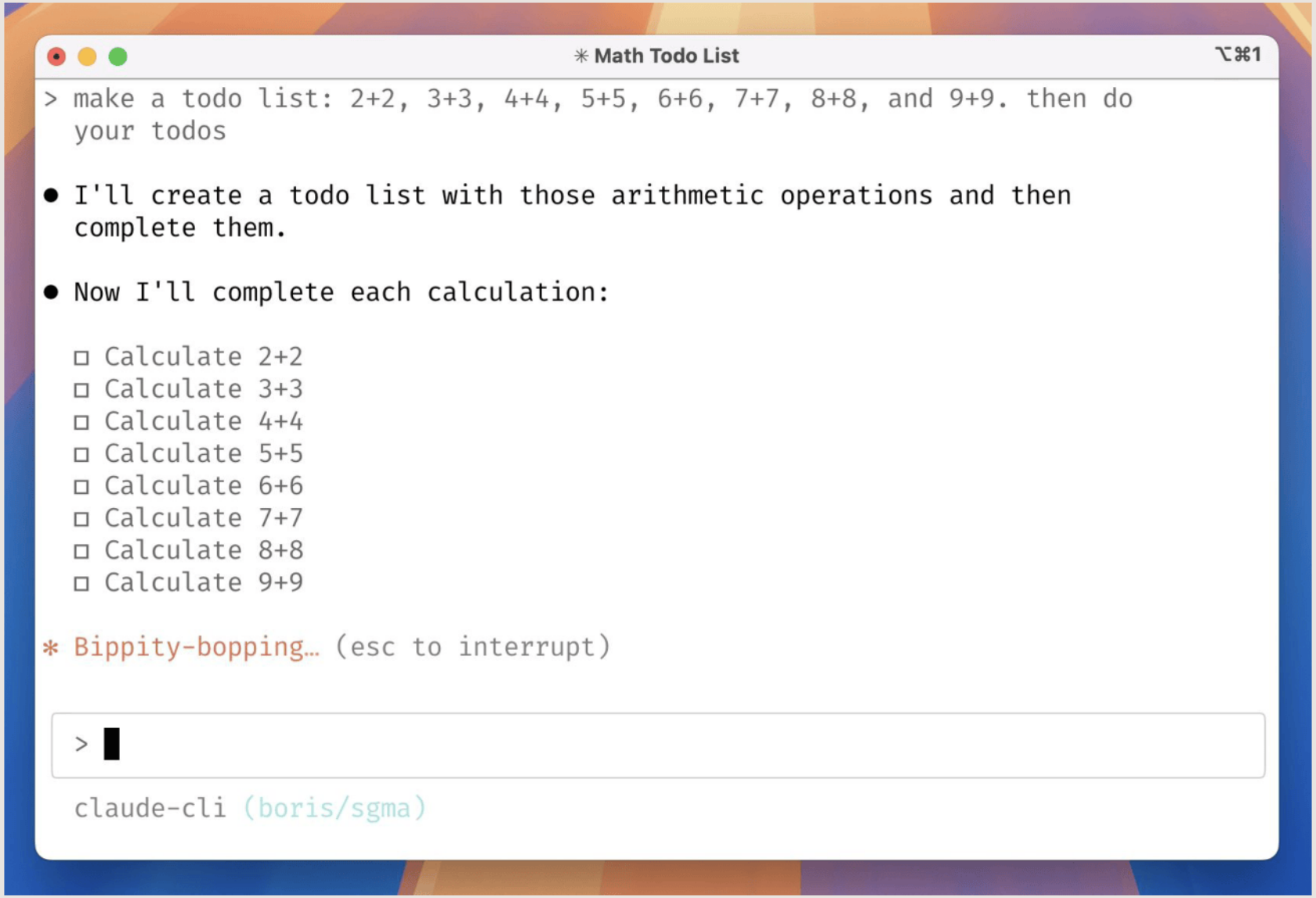
> actually don't show a todo list at all, and instead render the tool uses inline, as bold headings when the model starts working on a todo. keep the "step 2 of 4" or whatever, and add middot /todo to see after in grey



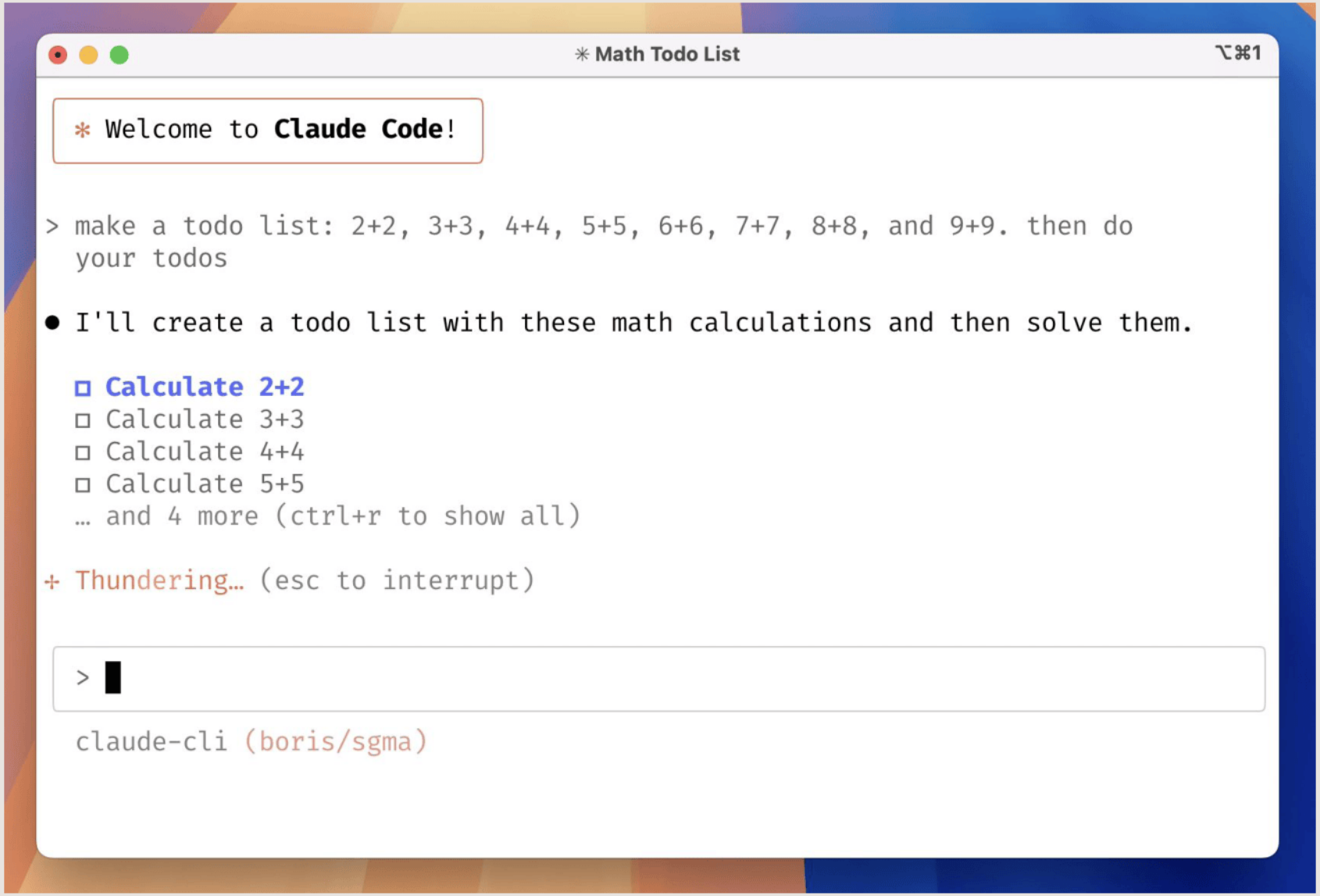
> also add a todo pill under the text input, similar to bg tasks. it should render "todos: 1 of 3" or whatever. make the pill interactive



> 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



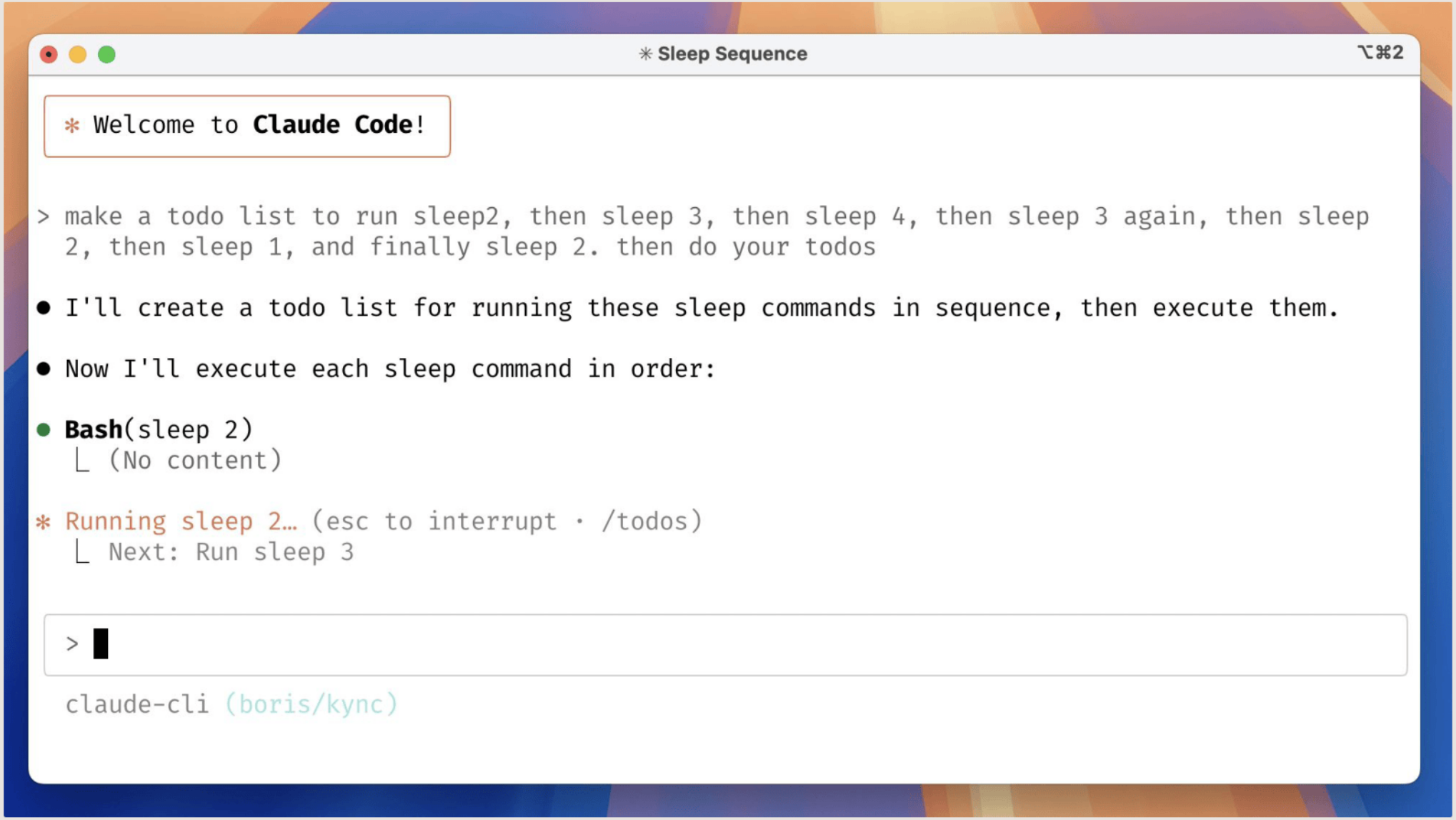
> actually what if you show the todo list above the input instead. truncate at 5 and show "... and 4 more" or whatever. add a heading "Todo:" in grey text



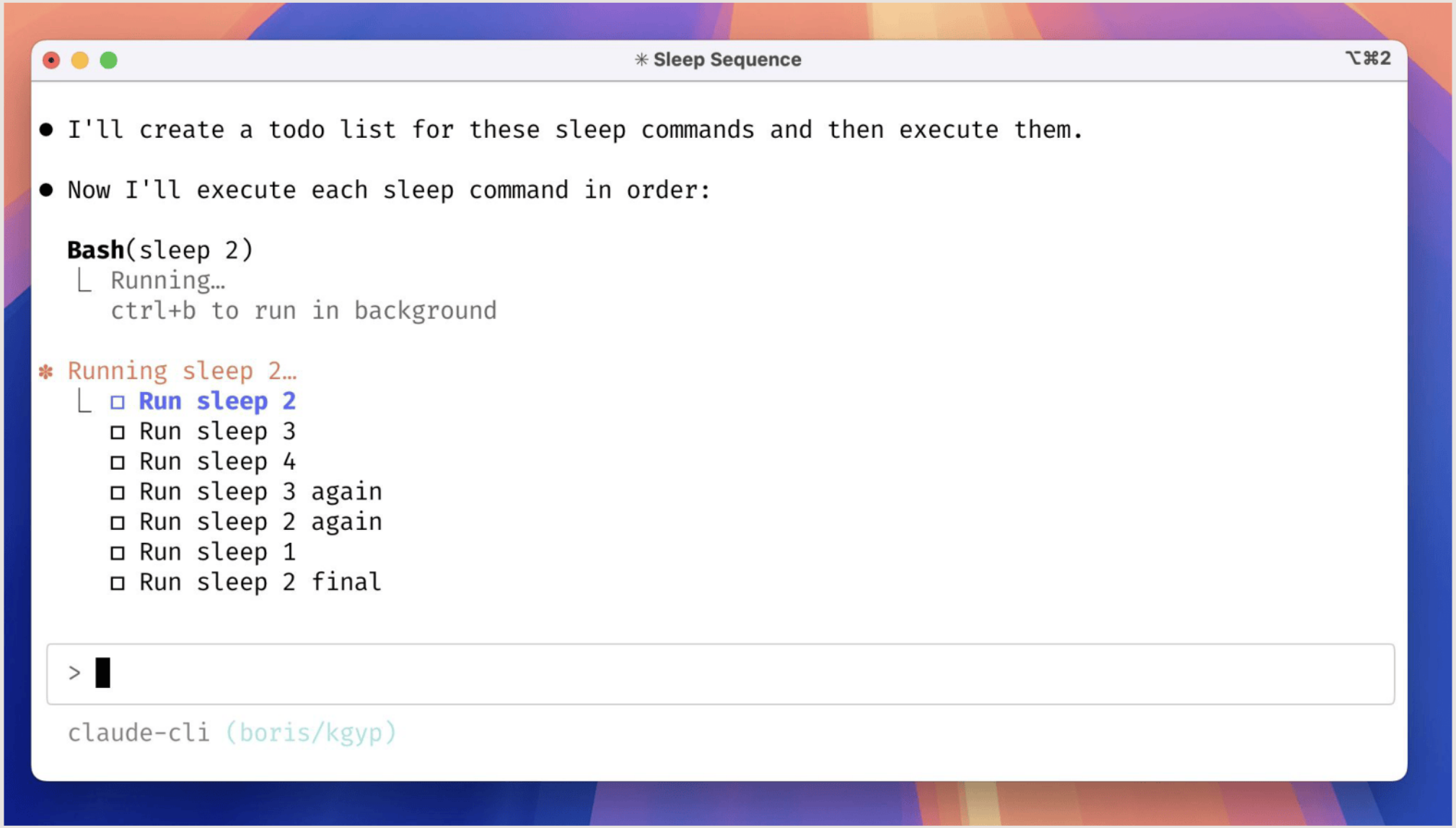
> actually what if you show the todo list above the input instead. truncate at 5 and show "... and 4 more" or whatever. add a heading "Todo:" in grey text



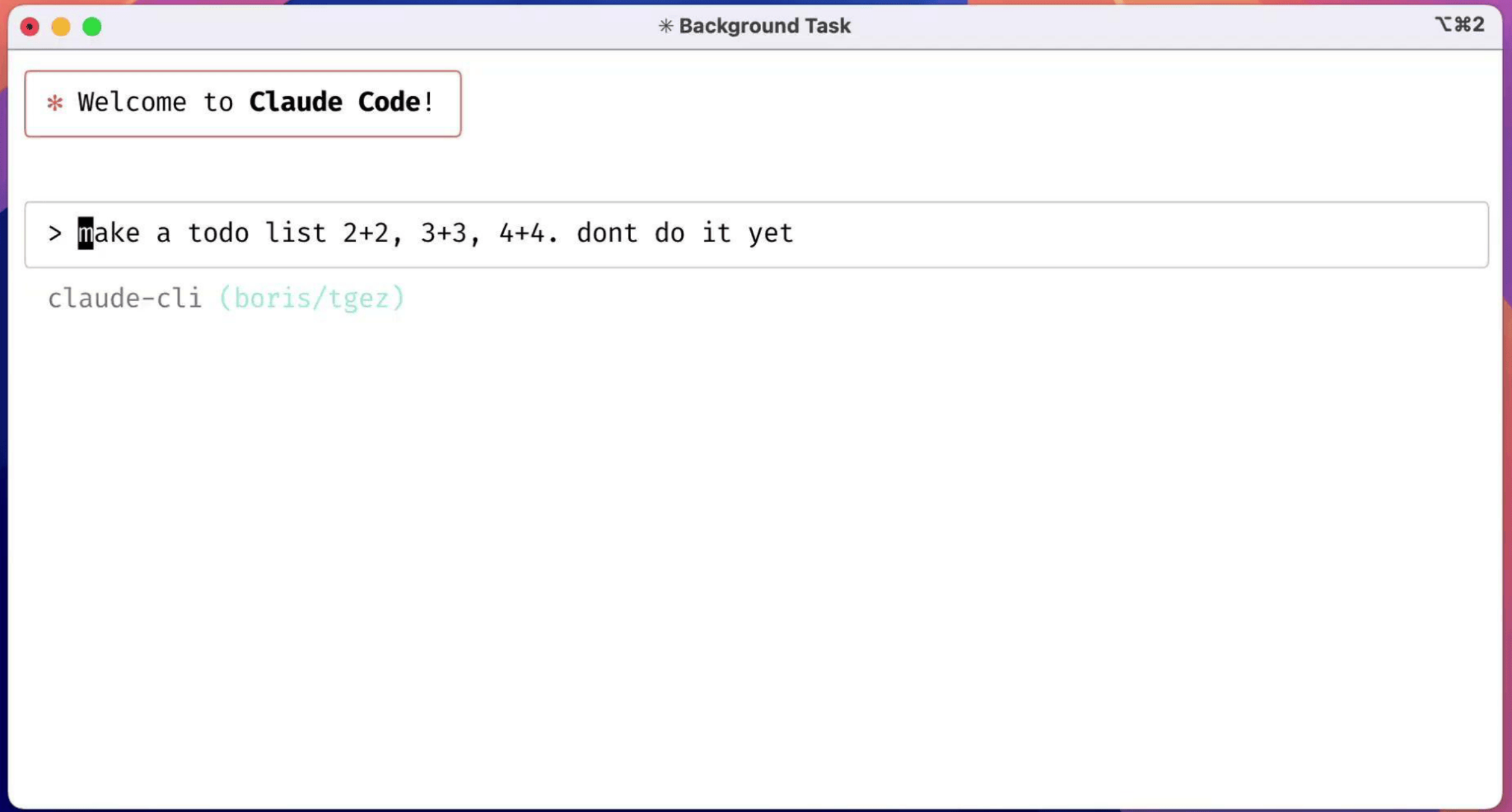
> actually what if you show the todo list above the input instead. truncate at 5 and show "... and 4 more" or whatever. add a heading "Todo:" in grey text



> instead of showing todos above the input, merge them into the spinner. show the current todo as the spinner message in active verb form, and the next todo instead of the spinner tip, in passive form. also add a /todos slash command to see all todos, and mention it next to "esc to interrupt"



> make it so i can hit ctrl+t to expand todos inline. also rm the "esc to interrupt" message when we show the todos, and replace it w "ctrl+t to expand"


✳ Background Task
⌥⌘2
✻ Welcome to Claude Code!
> make a todo list 2+2, 3+3, 4+4. dont do it yet
claude-cli (boris/tgez)