

OpenAI

# Codex 301 - Advanced Features & Automations

How to fully leverage Codex for your team



Name goes here

Title goes here



- 01 Advanced Features
- 02 Long Horizon Tasks
- 03 Harness Engineering
- 04 SDK and Codex Exec
- 05 Codex Architecture

# Advanced Features



# Advanced features at a glance

1. Permissions
2. Sub-agents
3. Plan mode
4. Personality
5. Web search options
6. Windows sandbox
7. Status line
8. Writable roots
9. ExecRules
10. Memory

# 01 Permissions

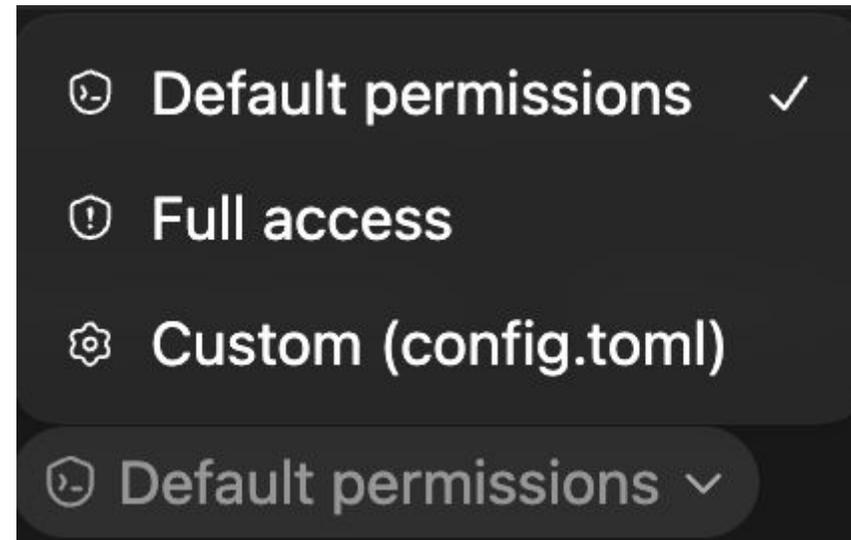
## Update Model Permissions

- 1. **Default** Codex can read and edit files in the current workspace, and run commands. Approval is required to access the internet or edit other files.
- 2. **Full Access** Codex can edit files outside this workspace and access the internet without asking for approval. Exercise caution when using.

**Default:** OnRequest approval policy + workspace-write sandbox. Blocks full-disk write access and keeps network very limited

**Full Access:** Never approval policy + danger-full-access sandbox. Full filesystem + network access

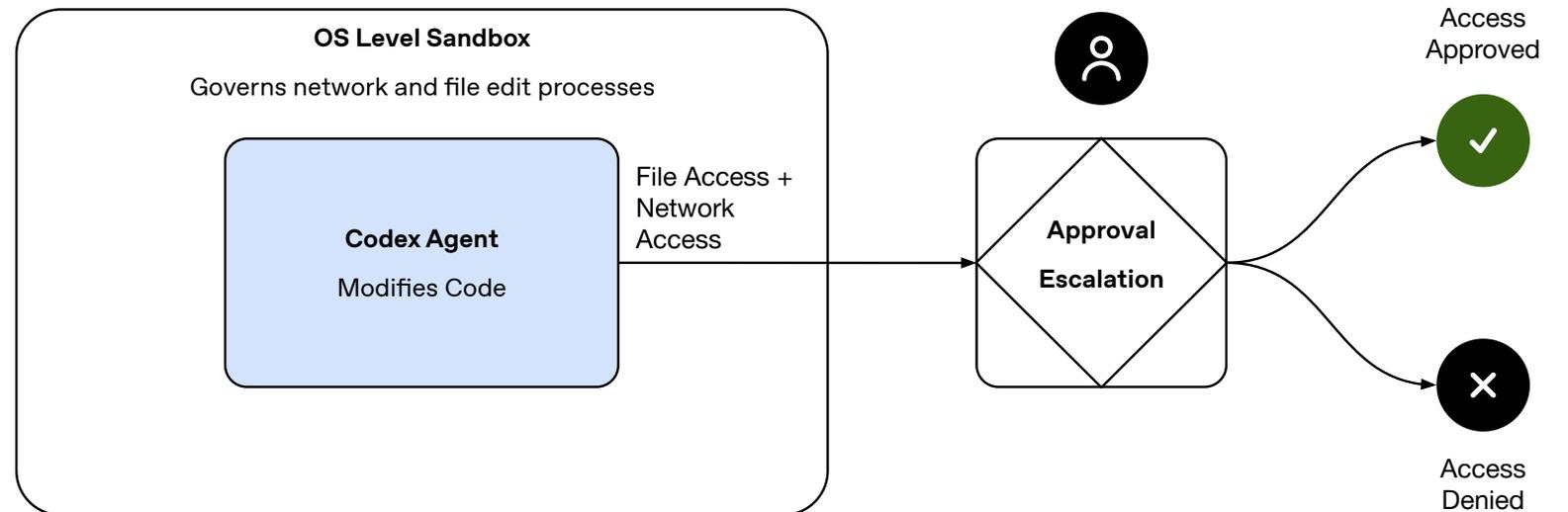
**Custom (config.toml):** Takes your custom approval + sandbox



# Built with OS-level sandboxing

OS-level sandboxing reduces risk by isolating the entire agent runtime, not just blocking individual tools and actions.

- **OS-Level file-system boundaries:** prevent the agent from accessing or deleting unauthorized files.
- **OS-Level network access protections:** prevent the agent from prompt injection via internet, prevent malicious software downloads.



# What's a Sandbox?

Codex can run commands, such as:

```
cat, cd, cut, echo, expr, false, grep, head
```

But we definitely want to block commands like:

```
rm -rf /, git reset --hard
```

Or connect to the internet by default:

```
curl external-website
```

It's also bound to your git working directory

```
printf "hi" > ../test.md # should ask for approval
```

Which makes Codex one of the safest AI SWE to use, but requires some customization for advance set-up

```
codex git:(main) x codex

>_ OpenAI Codex (v0.66.0)

model:      gpt-5.1-codex-max medium  /model to change
directory:  ~/code/codex

Tip: Use /feedback to send logs to the maintainers when something looks off.

> write a file in ~/.codex/test-pap.md with 'hi' in it

• The workspace allows writing files under /Users/pap/.codex without needing escalation, so I'll write the file there using a simple shell command like cat.

• Ran printf 'hi'
  | ' > /Users/pap/.codex/test-pap.md
  | (no output)

— Worked for 6s —

• Wrote hi to /Users/pap/.codex/test-pap.md.

> remove it now

• Proceeding to delete the specified test file from the sandbox workspace as it is allowed and writable.

• Ran rm /Users/pap/.codex/test-pap.md
  | (no output)

— Worked for 4s —

• Removed /Users/pap/.codex/test-pap.md.
Token usage: total=8,791 input=8,661 (+ 24,704 cached) output=130
To continue this session, run codex resume 019b0367-0585-7352-8e38-e51a39ebca97
```

# Codex is secure by default: approval + sandboxing

## approval\_mode

(When does Codex ask for your permission to run a command?)

1. **untrusted:** Request permission when it attempts to use a command that's not on its "trusted" list
2. **on-failure:** If command fails, will ask to run outside sandbox.
3. (Default) **on-request:** Runs until it decides that it needs to ask for escalated permissions
4. **never:** Never ask for approvals

## sandbox\_mode

(Can Codex read / write in the directory, and which files can it access?)

1. **read-only:** Requires permission if attempt to write to file
2. (Default) **workspace-write:** Write to files in current directory
3. **danger-full-access:** Disable sandboxing altogether

- **Sandbox mode limits access:** By default, Codex can only edit the current workspace and cannot use the network
- **Approval mode gates actions:** Codex pauses when it needs elevated permissions, so you stay in control
- **Escalate intentionally:** Loosen these settings only for trusted repos or specific tasks

Reference: <https://developers.openai.com/codex/security>

# 02 Sub-agents

**How it works:** Codex creates a separate agent thread with inherited context/config, runs it independently

**Why this exists:** it lets one main thread fan out work (e.g., research + synthesis) without blocking the primary turn

Add `multi_agent = true` to `config.toml`

```
• Agent spawned
  ↳ call: call_wFNkY1BaExuS6Toj14veLU0v
    agent: 019c6782-f41b-75e0-a9d4-92c3b6fc2331
    status: pending init
    prompt: Find the N+1 source for ticket table rendering in /Users/derri
ckchoi/Code/agent-desk.
Focus on components/ticket-table.tsx and related data fetch logic. Iden...

• Agent spawned
  ↳ call: call_bgIF714b1iC68H7qvE5mGEa0
    agent: 019c6782-f8c4-7822-b045-9868f5763d93
    status: pending init
    prompt: Prepare a concrete patch plan for fixing the N+1 issue reporte
d in components/ticket-
table.tsx. You should inspect required files in the repo and propose min..
```

```
• Waiting for agents
  ↳ call: call_QFcNkqZi1IzkuZiH5IFIS2o2
    receivers: 019c6782-f41b-75e0-a9d4-92c3b6fc2331, 019c6782-f8c4-7822-b0
45-9868f5763d93

• Wait complete
  ↳ call: call_QFcNkqZi1IzkuZiH5IFIS2o2
    agents: 1 total · 1 completed
    019c6782-f8c4-7822-b045-9868f5763d93 completed: `components/ticket-tab
```

# 02 Multi-agents

- Enable multi-agents in the Codex settings  
~/`.codex/config.toml` file or `/experimental`
- Each agent has a description, distinct developer instruction, and model + reasoning settings, and sandboxing
  - Built-in roles: `default`, `worker`, `explorer`, `monitor`
- E.g. `reviewer`

```
model = "gpt-5.3-codex"  
model_reasoning_effort = "high"  
developer_instructions = "Focus on high priority  
issues, write tests to validate hypothesis before  
flagging an issue. When finding security issues give  
concrete steps on how to reproduce the vulnerability."
```



```
[features]  
multi_agent = true
```



```
[agents.default]  
description = "General-purpose helper."  
  
[agents.reviewer]  
description = "Find security, correctness, and test risks  
in code."  
config_file = "agents/reviewer.toml"  
  
[agents.explorer]  
description = "Fast codebase explorer for read-heavy  
tasks."  
config_file = "agents/custom-explorer.toml"
```

# 02 Multi-agents

- ✓ Define investigation slices and assign them to multiple explorer agents
- Launch awaiter agents to monitor any long-running repo inspection commands
- Synthesize agent findings into a codebase map and key follow-up areas

The agent fan-out is running now. I'm waiting on all six in parallel so we can synthesize once the first wave of findings is complete instead of polling them one by one.

## • Waiting for 6 agents

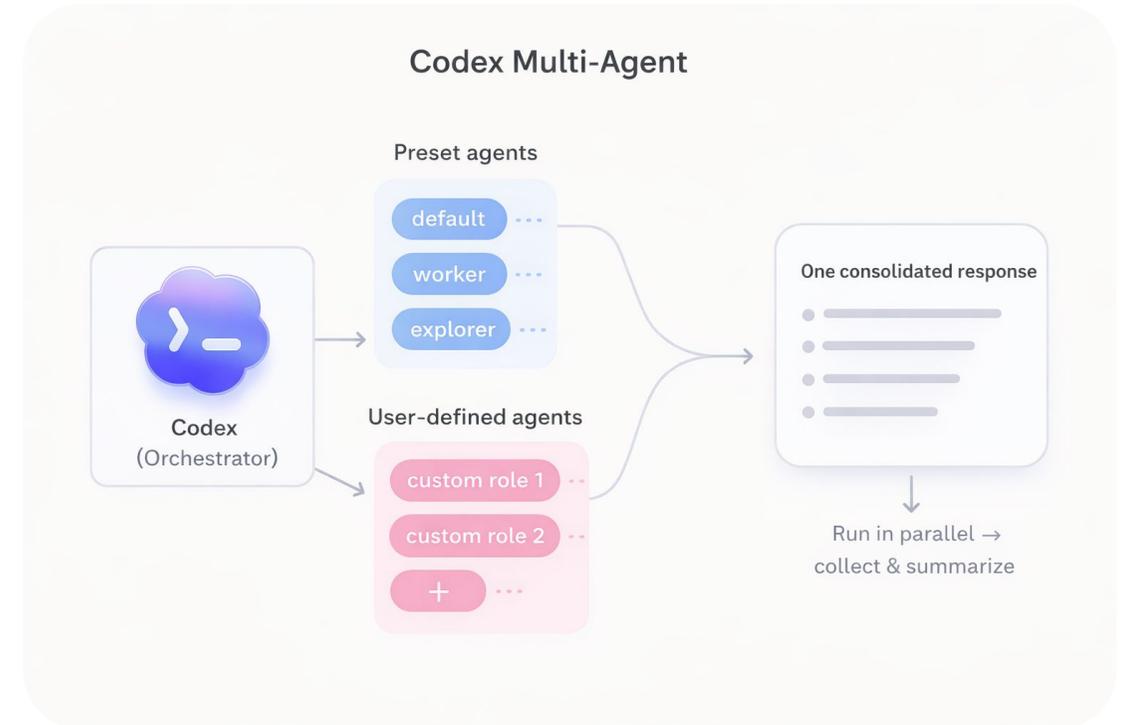
- └ Sage [explorer]
- └ Spruce [explorer]
- └ Yew [explorer]
- └ Camellia [explorer]
- └ Ash [awaiter]
- └ Date [awaiter]

## Agents

Select a thread to focus

- > 1. • Main [default] (current) 019c954b-fdda-7d92-a103-3e21176cc5c1
- 2. • Sage [explorer] 019c9551-d2c0-7892-b4af-8759f4edc30f
- 3. • Spruce [explorer] 019c9551-d3b2-7e11-aaf8-1d93cf7ad6ce
- 4. • Yew [explorer] 019c9551-d470-74f1-bca9-1b6a9dbc5bbd
- 5. • Camellia [explorer] 019c9551-d510-72f3-bfa7-db33d4003d38
- 6. • Ash [awaiter] 019c9551-d5e3-73a0-9b8e-ddd1f54a2d9f
- 7. • Date [awaiter] 019c9551-d699-7d53-8455-fcc00179f37e

Press enter to confirm or esc to go back



# 03 Plan Mode

`/plan` breaks task into short execution plan, then ask to approve before it starts making tool calls or file changes

Please create a second lock button in the composer UI.

I'll inspect the current composer internal footer layout first, then I'll draft a decision on where to place the second lock button.

Explored 1 search, 1 list

I found an existing `SandboxButton` in the composer internal footer. Next I'm reading those files to map out the current layout without duplicating state logic.

Explored 2 files

I've confirmed there's currently one `SandboxButton` rendered in the `ComposerInternalFooter` only. I'll check if other composer footers are selected so I can determine where the second button should live.

Explored 2 files, 1 search

I confirmed external and internal footer components. The internal one has the lock control. I'll check if there are any other footers to shape a precise plan including tool calls.

Explored 3 files

I have enough code context to draft a plan. I'll list choices that materially affect it (placement, behavior). The plan is implementation-ready, ready for your approval.

Asking questions

```
• I found the sandbox-related components; next I'm reading sandbox-button and composer footer/layout files to determine exactly where the existing control appears and what "another" can mean in implementation terms.
```

```
• Explored  
  ↳ Read sandbox-button.tsx, composer-external-footer.tsx, composer-internal-footer.tsx, composer.tsx
```

```
• I confirmed there is currently a single SandboxButton, rendered only in the right side of ComposerInternalFooter and only when !isCollabControl. I'm now checking related stories/feature flags to propose exact placement and behavior options for the second button.
```

```
• Explored  
  ↳ Read sandbox-button.stories.tsx, use-collaboration-mode-enabled.ts  
  ↳ Search isCollabControl|useCollaborationModeEnabled|collab in composer-internal-footer.tsx
```

```
• I've narrowed it down: the current button is intentionally hidden for collaboration-control variants, so adding "another" likely means a second placement that remains visible in those layouts. I'll quickly check existing footer stories/tests so the plan includes concrete validation coverage.
```

```
• Explored  
  ↳ Search ComposerInternalFooter|composer-internal-footer|SandboxButton in composer  
  ↳ Read composer.stories.tsx  
  ↳ List composer  
  ↳ Search footer|sandbox|composer.*stories  
  ↳ Read composer-footer.css
```

Question 1/3 (3 unanswered)  
Where should the additional `sandbox` lock button live in the composer UI?

- > 1. Mode row (Recommended) Add it beside mode/model controls in the left footer group so `sandbox` state stays visible with mode selection. Place a second copy near dictation/send controls (next to the existing lock area).
- 2. Right action group
- 3. External footer row Place it in the top/external footer near local/worktree environment controls.
- 4. None of the above Optionally, add details in notes (tab).

`tab` to add notes | enter to submit answer | ctrl + n next question | esc to interrupt

Where should the second sandbox lock button appear? < 1 of 3 >

1. External footer right
2. External footer left
3. Both sides
4. No, and tell Codex what to do differently

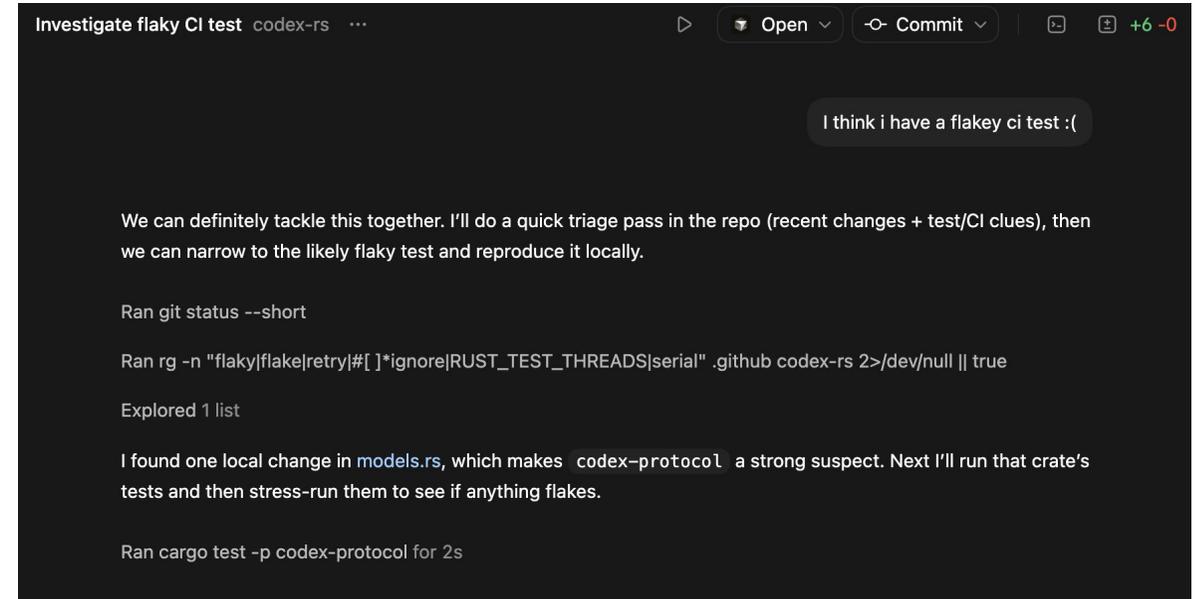
Dismiss `ESC` Continue `enter`

# 04 Personality

`/personality` controls behavior like brevity, formality, and explanation style while keeping the same tools and permissions behind the scenes.

## Options

1. Friendly - Warm, collaborative, and helpful.
2. Pragmatic - Concise, task-focused, and direct



```
Investigate flaky CI test codex-rs ...
I think i have a flakey ci test :(

We can definitely tackle this together. I'll do a quick triage pass in the repo (recent changes + test/CI clues), then we can narrow to the likely flaky test and reproduce it locally.

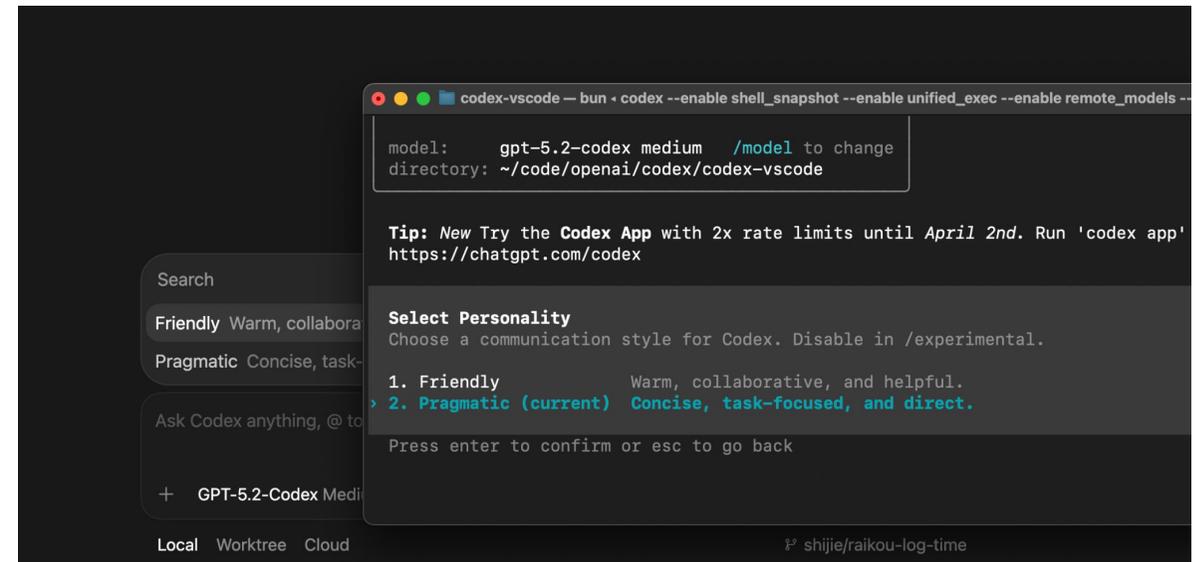
Ran git status --short

Ran rg -n "flake|flake|retry|#[*]ignore|RUST_TEST_THREADS|serial" .github codex-rs 2>/dev/null || true

Explored 1 list

I found one local change in models.rs, which makes codex-protocol a strong suspect. Next I'll run that crate's tests and then stress-run them to see if anything flakes.

Ran cargo test -p codex-protocol for 2s
```



```
codex-vscode — bun • codex --enable shell_snapshot --enable unified_exec --enable remote_models --
model: gpt-5.2-codex medium /model to change
directory: ~/code/openai/codex/codex-vscode

Tip: New Try the Codex App with 2x rate limits until April 2nd. Run 'codex app'
https://chatgpt.com/codex

Search
Friendly Warm, collabora
Pragmatic Concise, task-
Ask Codex anything, @ to
+ GPT-5.2-Codex Medi

Select Personality
Choose a communication style for Codex. Disable in /experimental.

1. Friendly Warm, collaborative, and helpful.
> 2. Pragmatic (current) Concise, task-focused, and direct.

Press enter to confirm or esc to go back

Local Worktree Cloud shijie/raikou-log-time
```

# 05 Memory

- Enable memory in the Codex settings  
`~/.codex/config.toml`
- Idle sessions in the background are consolidated into a local memory workspace.
- Memories then influences future sessions by injecting a compact summary and making deeper memory files available for lookup.
- Codex can better carry forward durable preferences, recurring project context, and prior decisions across sessions.



```
[features]  
memories = true
```



```
~/.codex/memories/memory_summary.md  
  
## User Profile  
This user uses Codex as an  
execution-heavy engineering workbench,  
not a brainstorming toy...
```

# 06 World's first Windows Agent Sandbox

Windows now has two sandbox options: **elevated** (default Windows sandbox path) and **unelevated** (legacy/non-admin).

Windows users get a popup to ask “Do you want to allow this app to make changes?” and adds to config

- [windows] sandbox="elevated|unelevated"

What's changed

- Core security model has been hardened
- Elevated mode uses dedicated sandbox identities/capability SIDs with stronger ACL isolation plus more robust setup/error handling and telemetry so failures are visible/diagnosable

```
Set up the Codex agent sandbox to protect your files and control  
<https://developers.openai.com/codex/windows>  
  
> 1. Set up default sandbox (requires Administrator permissions)  
2. Use non-admin sandbox (higher risk if prompt injected)  
3. Quit  
  
Press enter to confirm or esc to go back
```

# 07 Status Line

**/statusline** configures which items to display in the status line

## Configure Status Line

Select which items to display in the status line.

Type to search

>

> [ ] <b>model-name</b>	<b>Current model name</b>
[ ] model-with-reasoning	Current model name with reasoning level
[ ] current-dir	Current working directory
[ ] project-root	Project root directory (omitted when unavailable)
[ ] git-branch	Current Git branch (omitted when unavailable)
[ ] context-remaining	Percentage of context window remaining (omitted when un...
[ ] context-used	Percentage of context window used (omitted when unknown)
[ ] five-hour-limit	Remaining usage on 5-hour usage limit (omitted when una...

# 08 Writable\_roots

Extend the path to which Codex can write,  
e.g:

```
1 [sandbox_workspace_write]
2 writable_roots = ["/Users/pap/.codex/"]
```

Useful for Gradle developers, iOS developers

Examples paths:

Python: ~/.cache/uv

Rust: ~/.cargo/registry

Go: ~/.cache/go-build

Java/Gradle/Maven: ~/.gradle/caches,  
~/.gradle/wrapper, ~/.m2/repository

Bazel/ccache: ~/.cache/bazel, ~/.ccache

*Note: Codex doesn't necessarily need to write here -  
reading doesn't require things to be added to  
writable\_roots*

```
codex git:(main) x codex

>_ OpenAI Codex (v0.66.0)

model:      gpt-5.1-codex-max medium  /model to change
directory:  ~/code/codex

Tip: Use /feedback to send logs to the maintainers when something looks off.

> write a file in ~/.codex/test-pap.md with 'hi' in it

• The workspace allows writing files under /Users/pap/.codex without needing escalation, so I'll write the file there using a simple shell command like cat.

• Ran printf 'hi'
  | ' > /Users/pap/.codex/test-pap.md
  | (no output)

- Worked for 6s -

• Wrote hi to /Users/pap/.codex/test-pap.md.

> remove it now

• Proceeding to delete the specified test file from the sandbox workspace as it is allowed and writable.

• Ran rm /Users/pap/.codex/test-pap.md
  | (no output)

- Worked for 4s -

• Removed /Users/pap/.codex/test-pap.md.
Token usage: total=8,791 input=8,661 (+ 24,704 cached) output=130
To continue this session, run codex resume 019b0367-0585-7352-8e38-e51a39ebca97
```

# 09 ExecRules

Control which shell commands Codex can execute, using rule-based allow / prompt / forbid decisions.

~/.codex/rules/default.rules →

✓ codex execpolicy check gh pr  
791

✗ codex execpolicy check gh pr delete  
712

```
1 # Prompt before running commands with the prefix `gh pr view` outside the sandbox.
2 prefix_rule(
3     # The prefix to match.
4     pattern=["gh", "pr", "view"],
5     # The action to take when Codex requests to run a matching command.
6     decision="prompt",
7     # Optional rationale for why this rule exists.
8     justification="Viewing PRs is allowed with approval",
9     # `match` and `not_match` are optional "inline unit tests" where you can
10    # provide examples of commands that should (or should not) match this rule.
11    match=[
12        "gh pr view 7888",
13        "gh pr view --repo openai/codex",
14        "gh pr view 7888 --json title,body,comments",
15    ],
16    not_match=[
17        # Does not match because the `pattern` must be an exact prefix.
18        "gh pr --repo openai/codex view 7888",
19    ],
20 )
```

# Long Horizon Tasks



# Long horizon tasks with Codex



Derrick Choi

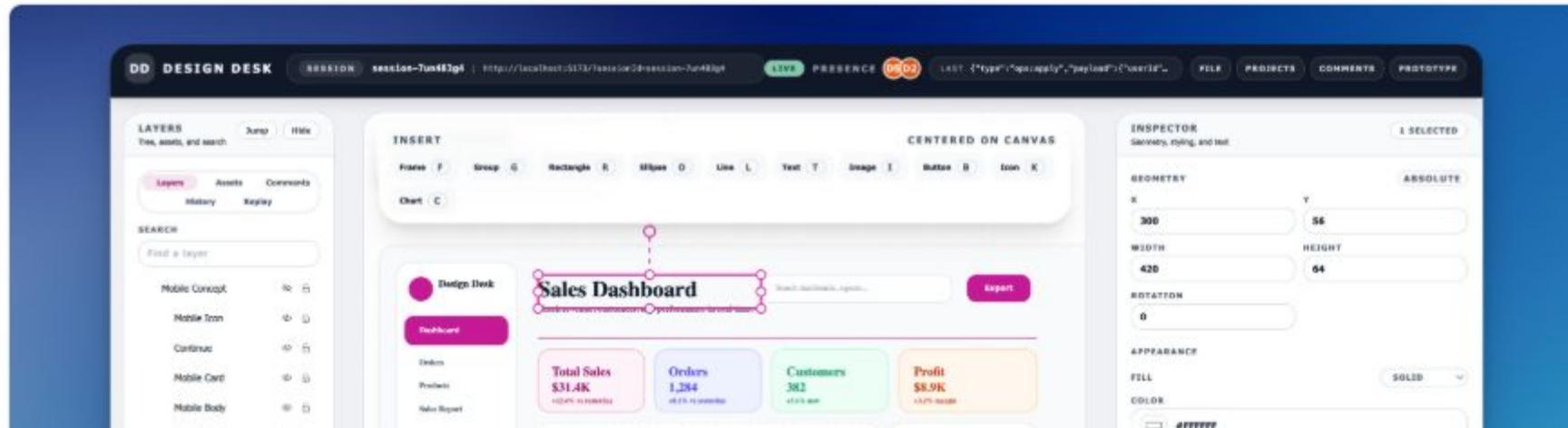
[View on GitHub](#)

[Download raw](#)

In September 2025, OpenAI introduced GPT-5-Codex as the first version of GPT-5 optimized for agentic coding. In December 2025, we launched 5.2 which was the moment that people began to believe that using autonomous coding agents could be reliable. In particular, we saw a huge jump in how long the model could reliably follow instructions.

I wanted to stress-test that threshold. So I gave Codex a blank repo, full access, and one job: build a design tool from scratch. Then I let it run with GPT-5.3-Codex at “Extra High” reasoning. Codex ran for about 25 hours uninterrupted, used about 13M tokens, and generated about 30k lines of code.

This was an experiment, not a production rollout. But it performed well on the parts that matter for long-horizon work: following the spec, staying on task, running verification, and repairing failures as it went.

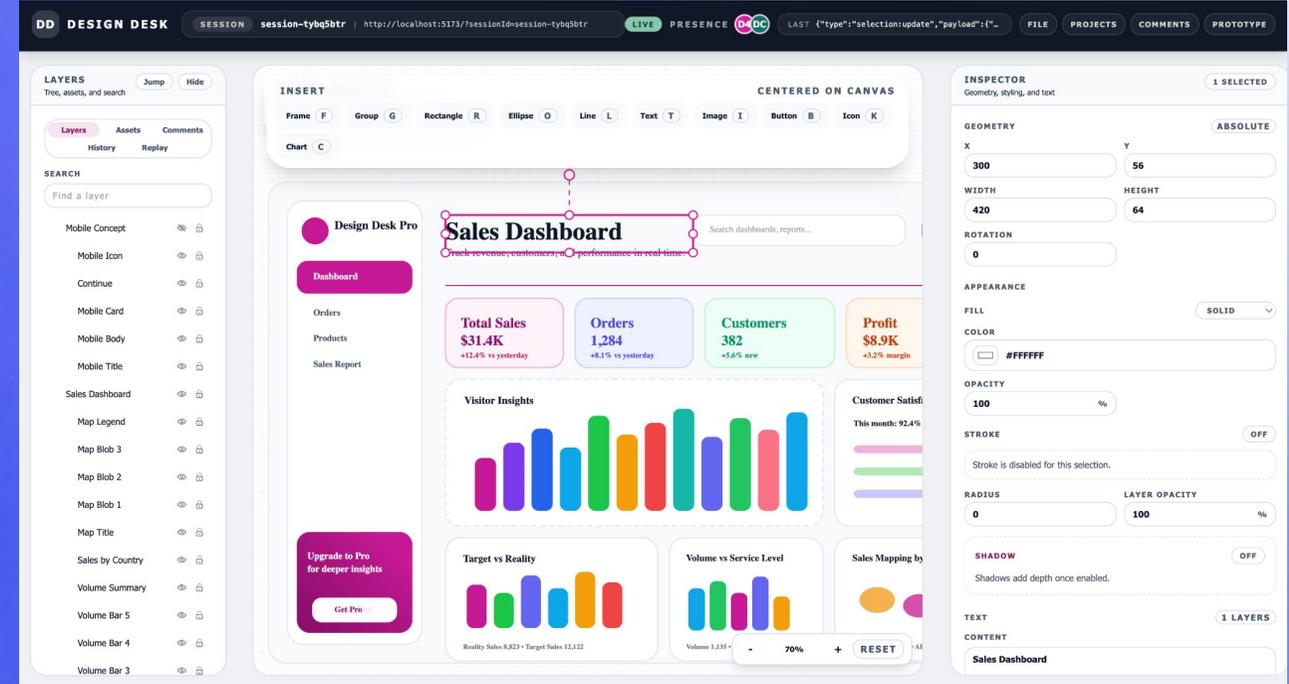


# How Codex ran for >24 hrs

## Steps (cookbook):

1. **Prompt.md** that contains the prompt with goals, requirements, deliverables, product spec
2. **Plans.md** that houses implementation plan with milestones and validation step. **Architecture.md** that has the guiding principles
3. **Implement.md** that contains the implement prompt that references Plans.md using GPT-5.2-Codex on X-High reasoning
4. **Documentation.md** where milestone status lives as Codex implements

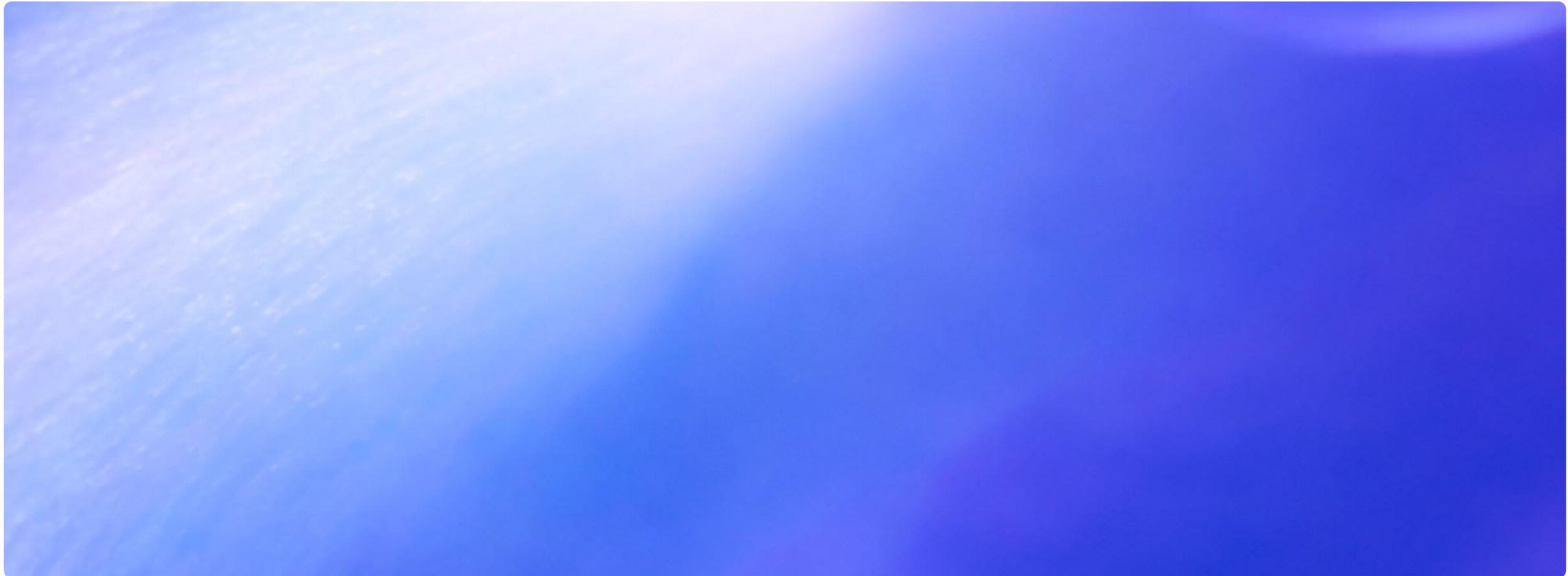
Result: 13M tokens, 25 hour execution, 45k LOC



Token usage: total=12,727,845 input=10,720,903 (+ 267,757,568 cached)

# Build Automations with Codex

OpenAI



# Codex SDK

Programmatic way to control Codex agent

- Same Codex behavior, but triggered from CI, scripts, or internal tools
- Turn repeatable tasks into an API: lint, refactors, migrations, test generation, release notes
- Keep guardrails on by default: approvals + sandboxing help teams ship automation safely
- Scale beyond the CLI: orchestrate larger workflows + wire results into existing systems

```
1 import { Codex } from "@openai/codex-sdk";
2
3 const codex = new Codex();
4
5 const thread = codex.startThread();
6
7 const result = await thread.run(
8   "Make a plan to diagnose and fix the CI
9   failures"
10 );
11 console.log(result);
```

# Embed Codex functionality using headless mode

## **Security Triage**

Codex can automatically review your code for security and quality issues during CI runs, ranking findings and even proposing verified patches

## **Refactor & Cleanup Automation**

Codex running as background jobs so your codebase stays clean without devs needing to manually sweep through it.

## **Test Coverage Bot**

Codex can detect uncovered files in your repo and generate unit tests right in the CI pipeline.

## **Release Hygiene Automation**

Automate the release chores like updating changelogs, refreshing READMEs or AGENTS.md, and scaffolding migration scripts.

# Analyze for code quality issues

```
codex exec "analyze this app for code quality issues." --output-schema codex-output-schema.json | jq
```

- Codex adheres to output JSON schema
- Summarize audit findings and return issues identified and recommendations

```
{
  "type": "object",
  "properties": {
    "summary": {
      "type": "object",
      "properties": {
        "total_files": {
          "type": "integer",
          "description": "Total number of files analyzed"
        },
        "total_issues": {
          "type": "integer",
          "description": "Total number of quality issues detected across all files"
        },
        "score": {
          "type": "number",
          "description": "Overall code quality score (0-100, higher is better)",
          "minimum": 0,
          "maximum": 100
        }
      }
    },
    "required": [
      "total_files",
      "total_issues",

```

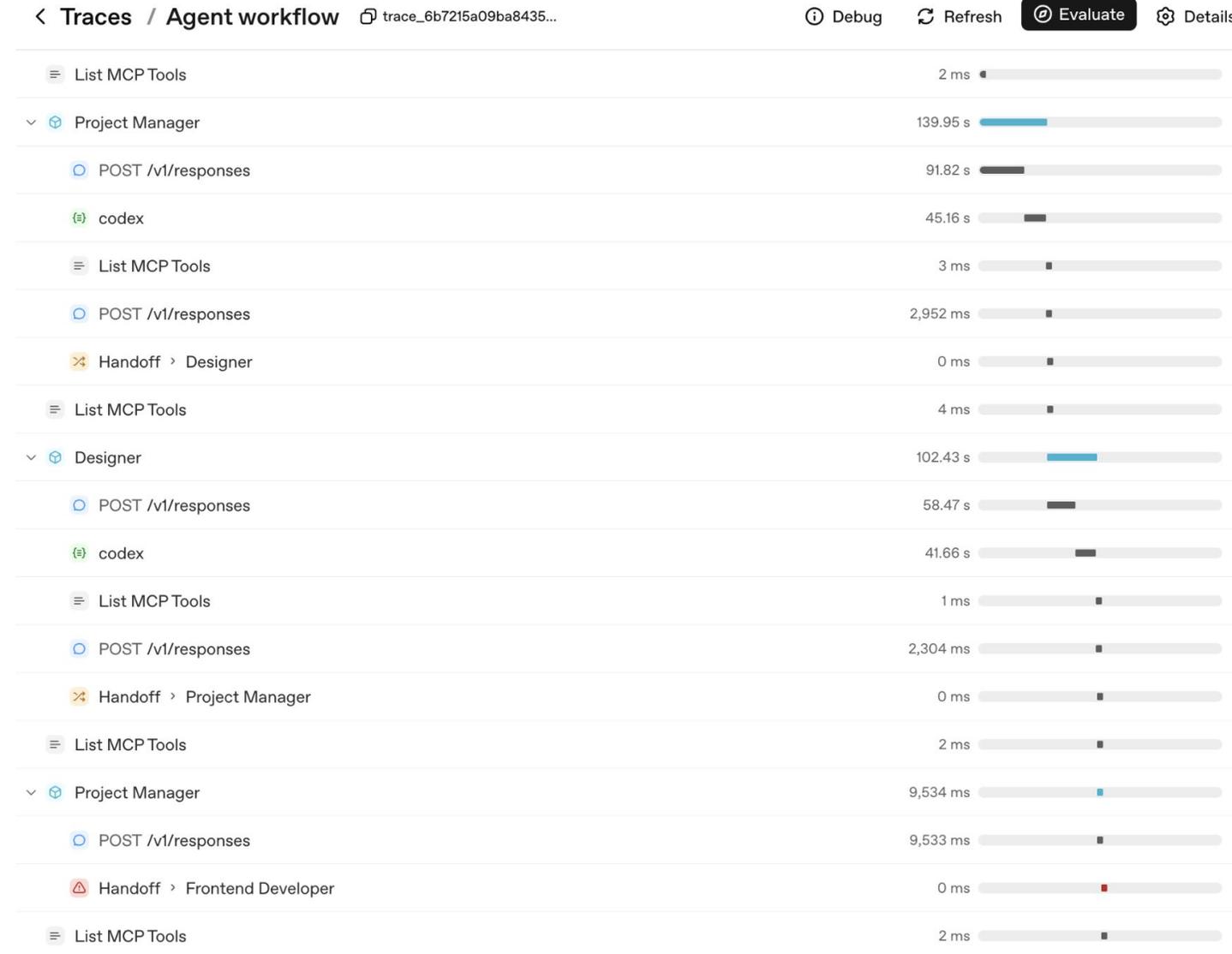
Schema

```
{
  "summary": {
    "total_files": 12,
    "total_issues": 3,
    "score": 65
  },
  "issues": [
    {
      "file": "package.json",
      "line": 5,
      "severity": "major",
      "type": "tooling",
      "description": "`npm run lint` is defined here but the command never finishes—it drops into the Next.js ESLint configuration prompt because the repo lacks any `.eslintrc`/config. I couldn't get automated lint output (the CLI stops at the prompt) so the project currently ships without any gatekeeping lint checks."
    },
    {
      "file": "package-lock.json",
      "line": 1,
      "severity": "major",
      "type": "dependency-management",
      "description": "Next.js warns about multiple lockfiles and ends up reading `/Users/derrickchoi/package-lock.json`; the repo also contains `pnpm-lock.yaml:1` yet no single lock is honored. Running tooling against the wrong lockfile leads to inconsistent installs, so the extra lockfile needs to be removed or ignored and the remaining manager (pnpm) explicitly used."
    },
    {
      "file": "pages/api/generate-agents.ts",
      "line": 49,
      "severity": "critical",
      "type": "security",
      "description": "The POST handler exposes the OpenAI key to anyone hitting `/api/generate-agents` (there's no authentication, rate limit, or origin guard), so an attacker can fire unlimited requests to `gpt-4.1` and exhaust the project's API quota / budget."
    }
  ],
  "recommendations": [
    {
      "area": "tooling",
      "recommendation": "Add an `.eslintrc.json` (or equivalent) with the Next.js recommended rules so `next lint` runs non-interactively, then fix the emitted issues and re-run lint as part of CI."
    },
    {
      "area": "dependency management",
      "recommendation": "Keep only one lockfile (e.g., delete the stray `package-lock.json` and rely on `pnpm-lock.yaml`) and make the choice explicit in documentation and tooling so commands like `npm run lint` do not pick the wrong lockfile."
    }
  ]
}
```

CLI Output

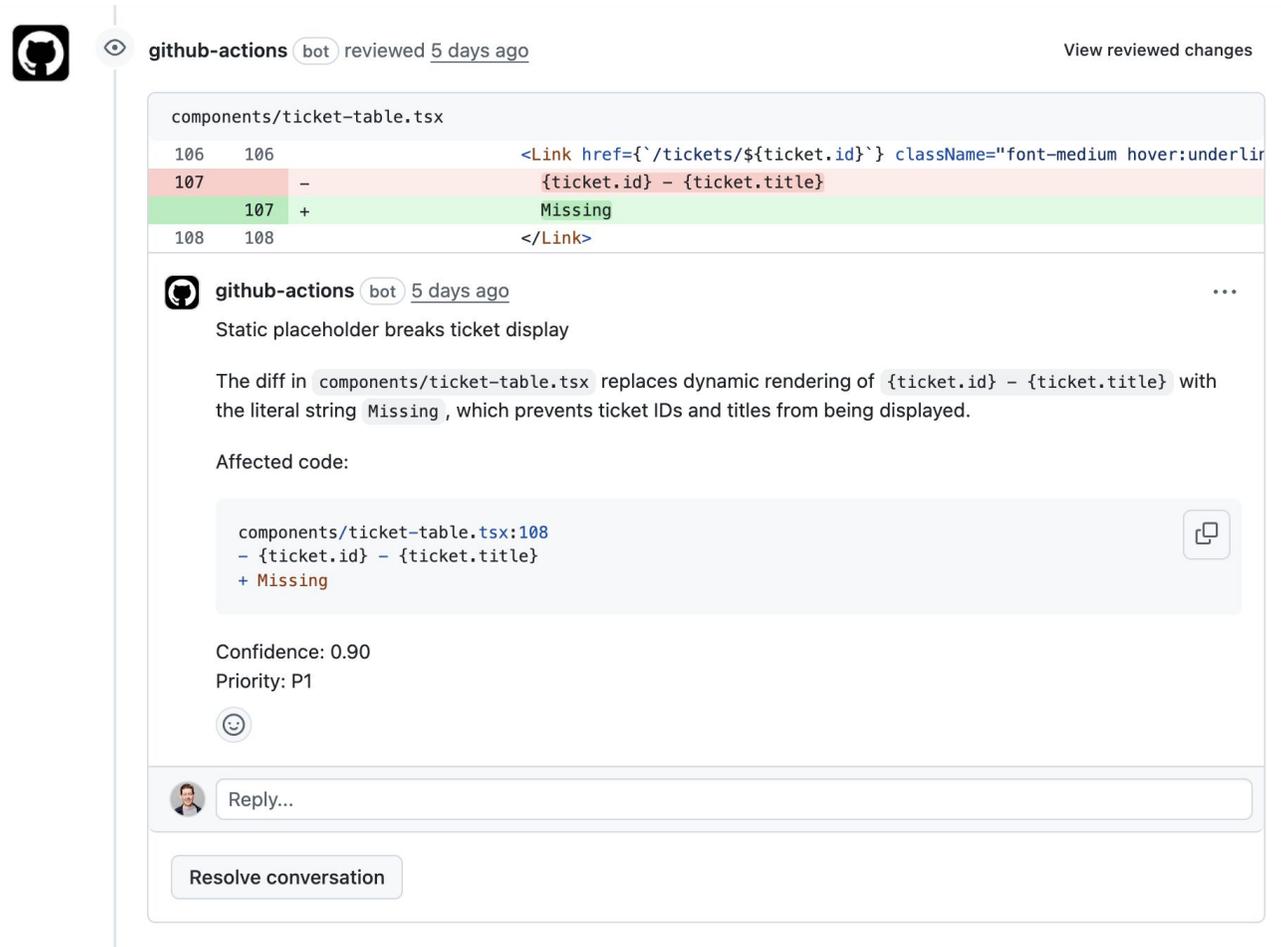
# Building Workflows with Codex & Agents SDK

- Use Codex as an MCP Server that can be called as a tool by an agent.
- Control context and handoffs between agents.
- Create agents designed for specific tasks: frontend, backend, test implementation.
- More examples in the OpenAI Cookbook:  
<https://cookbook.openai.com/topics/codex>



# On-Prem Code Review

- Use the Codex CLI in exec mode with the same system prompt we use for Code Review.
- Use GPT-5-Codex's unique Code Review capabilities.
- CLI Structured outputs produce a comment, priority, and citation.
- Runs in your on-prem CI/CD.



The screenshot shows a GitHub Actions bot review comment on a code diff. The diff is for the file `components/ticket-table.tsx`. The code changes are as follows:

```
106 106 <Link href={`/tickets/${ticket.id}`} className="font-medium hover:underlin
107 - {ticket.id} - {ticket.title}
107 + Missing
108 108 </Link>
```

The bot's comment, posted 5 days ago, reads: "Static placeholder breaks ticket display". It explains that the diff replaces the dynamic rendering of `{ticket.id} - {ticket.title}` with the literal string `Missing`, which prevents ticket IDs and titles from being displayed. The bot also provides the affected code snippet:

```
components/ticket-table.tsx:108
- {ticket.id} - {ticket.title}
+ Missing
```

The bot's confidence is 0.90 and the priority is P1. At the bottom of the comment, there is a "Reply..." input field and a "Resolve conversation" button.

# Auto-Fix Failing CI

- Trigger Codex based on CI failures.
- Codex checks out its own branch and works to fix the failures.
- Result: a PR that passes tests.

The screenshot shows a GitHub Pull Request interface. At the top, the navigation bar includes 'Code', 'Issues', 'Pull requests 1', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. The PR title is 'Auto-fix failing CI via Codex #1'. Below the title, it says 'github-actions bot wants to merge 1 commit into main from codex/auto-fix-'. A green 'Open' button is visible. The PR details show 'Conversation 0', 'Commits 1', 'Checks 0', and 'Files changed 2' with a diff of '+30 -2'. A comment from the 'github-actions bot' states: 'Codex automatically generated this PR in response to a CI failure on workflow Python CI Showcase . Failed run: https://github.com/oaiaicorp/codex-ci/actions/runs/17933707644 Head branch: main This PR contains minimal changes intended solely to make the CI pass.' Below the comment, the PR title is repeated: 'fix(ci): auto-fix failing tests via Codex' with commit hash '2ff47e5'. A green checkmark indicates 'No conflicts with base branch' and 'Merging can be performed automatically.' A 'Merge pull request' button is present. The right sidebar shows 'Reviewers' (himadri-openai), 'Assignees' (No one), 'Labels' (None yet), 'Projects' (None yet), 'Milestone' (No milestone), and 'Development' (Successfully merging this pull request may close these issues. None yet).

# Automatically Label Issues

- Trigger Codex on issue creation.
- Categorize issue based on a series of predefined labels.
- Update issues with the relevant label.

```
- id: codex
uses: openai/codex-action@main
with:
  openai-api-key: ${ secrets.CODEX_OPENAI_API_KEY }
  allow-users: "*"
  prompt: |
    You are an assistant that reviews GitHub issues for the repository.

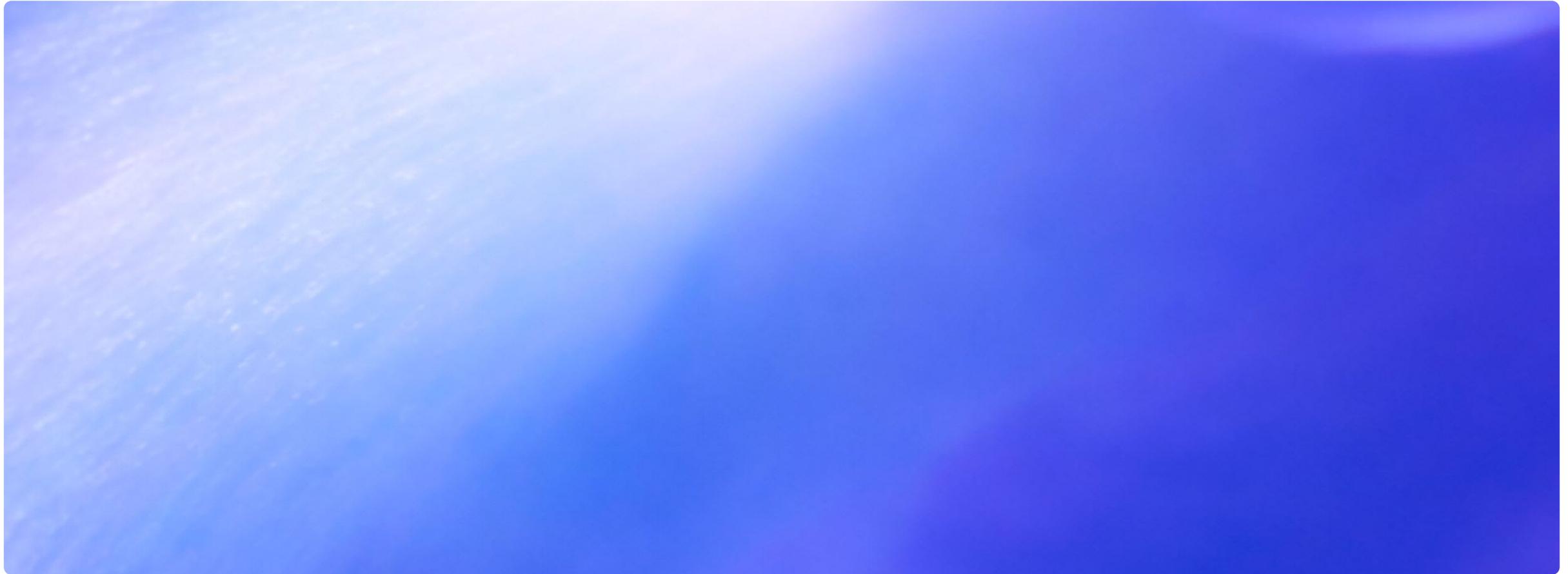
    Your job is to choose the most appropriate existing labels for the issue described later in this prompt.
    Follow these rules:
    - Only pick labels out of the list below.
    - Prefer a small set of precise labels over many broad ones.

    Labels to apply:
    1. bug - Reproducible defects in Codex products (CLI, VS Code extension, web, auth).
    2. enhancement - Feature requests or usability improvements that ask for new capabilities, better ergonomics,
    3. extension - VS Code (or other IDE) extension-specific issues.
    4. windows-os - Bugs or friction specific to Windows environments (always when PowerShell is mentioned, path
    5. mcp - Topics involving Model Context Protocol servers/clients.
    6. codex-web - Issues targeting the Codex web UI/Cloud experience.
    8. azure - Problems or requests tied to Azure OpenAI deployments.
    9. documentation - Updates or corrections needed in docs/README/config references (broken links, missing exam
    10. model-behavior - Undesirable LLM behavior: forgetting goals, refusing work, hallucinating environment det

    Issue number: ${ github.event.issue.number }
```

# Harness Engineering

OpenAI



# Let's dive into how this OpenAI team uses Codex

February 11, 2026 Engineering

## Harness engineering: leveraging Codex in an agent-first world

By Ryan Lopopolo, Member of the Technical Staff

 Listen to article | 18:04

 Share

Over the past five months, our team has been running an experiment: building and shipping an internal beta of a software product with **0 lines of manually-written code**.

The product has internal daily users and external alpha testers. It ships, deploys, breaks, and gets fixed. What's different is that every line of code—application logic, tests, CI configuration, documentation, observability, and internal tooling—has been written by Codex. We estimate that we built this in about 1/10th the time it would have taken to write the code by hand.

**Humans steer. Agents execute.**

We intentionally chose this constraint so we would build what was necessary to increase engineering velocity by orders of magnitude. We had weeks to ship what ended up being a

# How our internal team uses Codex (the extreme version)

This team ran a 5 month experiment: ship an internal beta product with **0 lines of manually-written code**

Codex wrote everything: product code, tests, CI, docs, observability, internal tooling

Result: roughly 1M lines of code, about 1,500 PRs merged

Core shift: engineering time moves from “typing code” to designing environments, intent, and feedback loops

**<https://openai.com/index/harness-engineering/>**

# The day-to-day loop (PRs, not prompts)

Prompt → agent run → PR → reviews → iterate

Humans interact via prompts, but the artifact is a pull request

Codex is instructed to: self-review locally, request additional agent reviews, address feedback, loop until satisfied

Agents use standard dev tools directly (gh, scripts, repo-embedded skills) to fetch context

We push review effort toward agent-to-agent review over time

# Make the product legible to the agent (UI + observability)

If the agent cannot see it, it cannot fix it

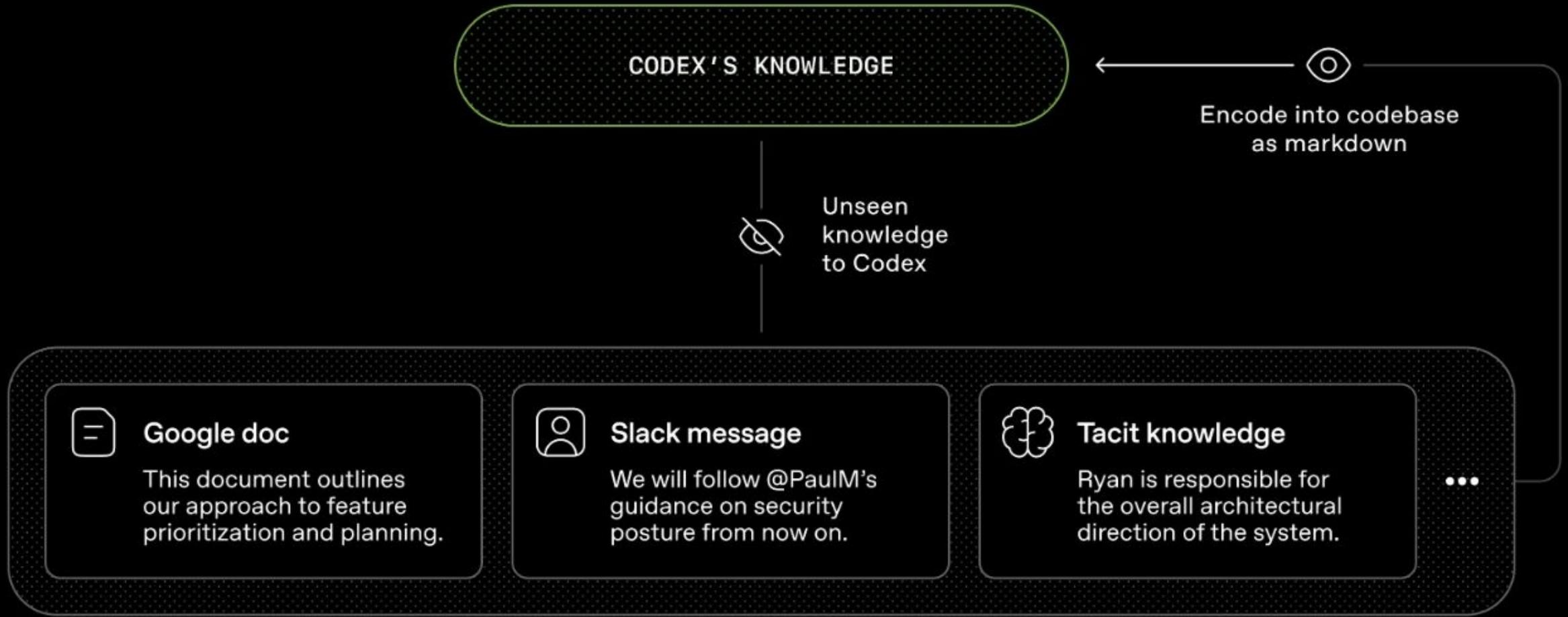
Our bottleneck became human QA, so we made the app itself legible to Codex

Worktree-per-change so Codex can boot and drive an instance per PR

Wired Chrome DevTools Protocol into the runtime and created skills for DOM snapshots, screenshots, navigation

Exposed logs, metrics, traces via an ephemeral observability stack per worktree; agents query with LogQL and PromQL

# The limits of agent knowledge: What Codex can't see doesn't exist



# Repository knowledge is the system of record (give a map, not a manual)

Keep context in the repo, versioned, and discoverable

Lesson: give Codex a map, not a 1,000 page instruction manual

Use a short AGENTS.md as a table of contents; deeper truth lives in structured docs/

Plans are first-class artifacts: active, completed, tech debt, decision logs, all in-repo

Enforce mechanically: linters and CI validate docs are fresh, cross-linked, structured, and agents open + fix PRs

```
1 AGENTS.md
2 ARCHITECTURE.md
3 docs/
4 |— design-docs/
5 |   |— index.md
6 |   |— core-beliefs.md
7 |   └─ ...
8 |— exec-plans/
9 |   |— active/
10 |   |— completed/
11 |   └─ tech-debt-tracker.md
12 |— generated/
13 |   └─ db-schema.md
14 |— product-specs/
15 |   |— index.md
16 |   |— new-user-onboarding.md
17 |   └─ ...
18 |— references/
19 |   |— design-system-reference-llms.txt
20 |   |— nixpacks-llms.txt
21 |   |— uv-llms.txt
22 |   └─ ...
23 |— DESIGN.md
24 |— FRONTEND.md
25 |— PLANS.md
26 |— PRODUCT_SENSE.md
```

Progressive disclosure: agents start with a small, stable entry point and are taught where to look next, rather than being overwhelmed up front

Architecture documentation provides a top-level map of domains and package layering

docs/ serves as repository's knowledge base

Design documentation includes verification status and a set of core beliefs that define agent-first operating principles

Plans are treated as first-class artifacts

# Guardrails that scale agent throughput

Rigid architectural model with validated dependency directions; enforced via custom linters and structural tests

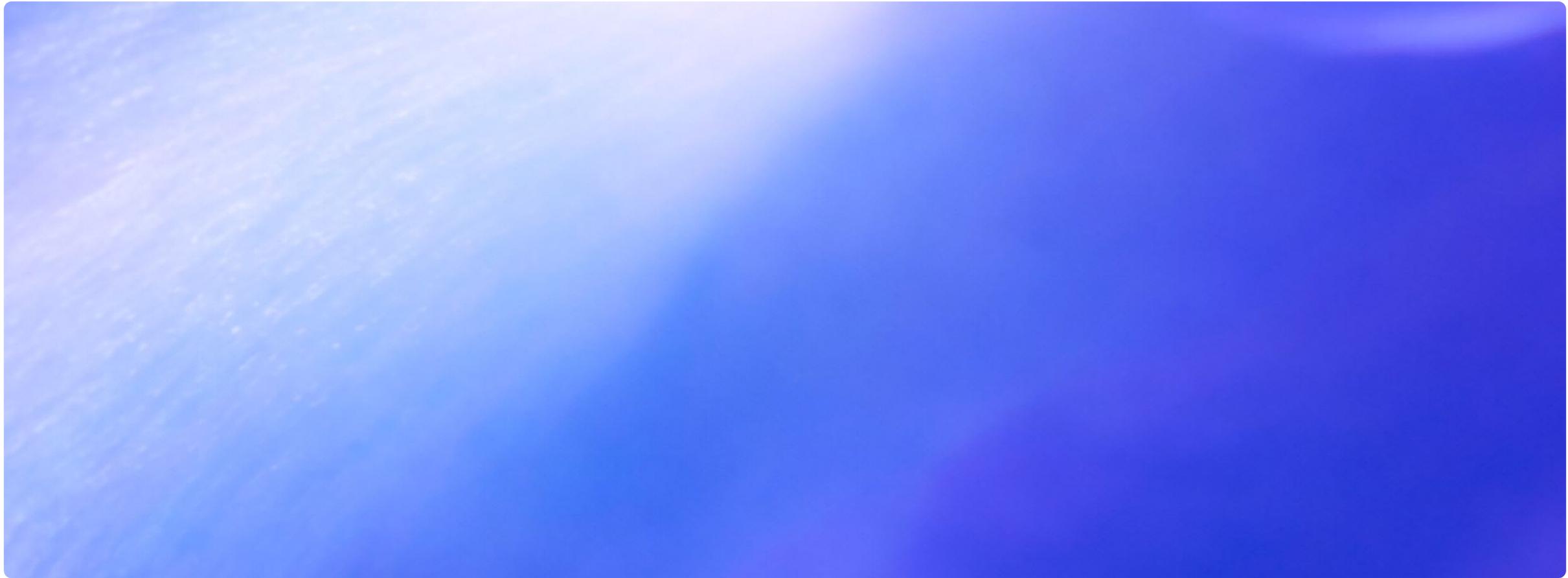
Encode “taste invariants” (logging, naming, file size limits, reliability rules) into tooling, not reviewer heroics

Merge philosophy shifts: minimal blocking gates, short-lived PRs, fixes are cheap, waiting is expensive

“Golden principles” + recurring cleanup tasks act like garbage collection for AI drift

What have you been using  
Codex for? Anything to  
highlight for the group?

# Codex Architecture



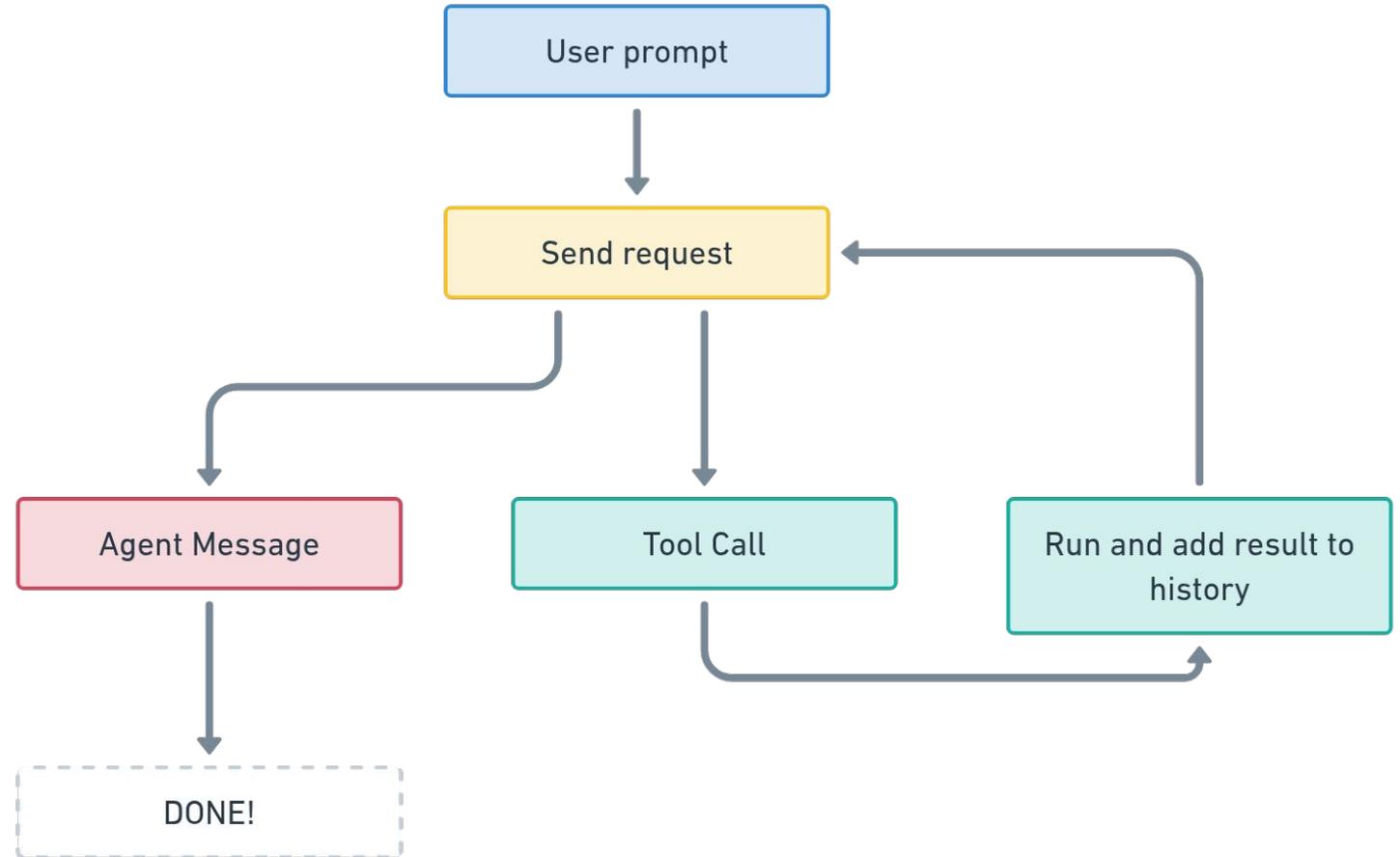
# Codex Architecture

OpenAI



# Agent

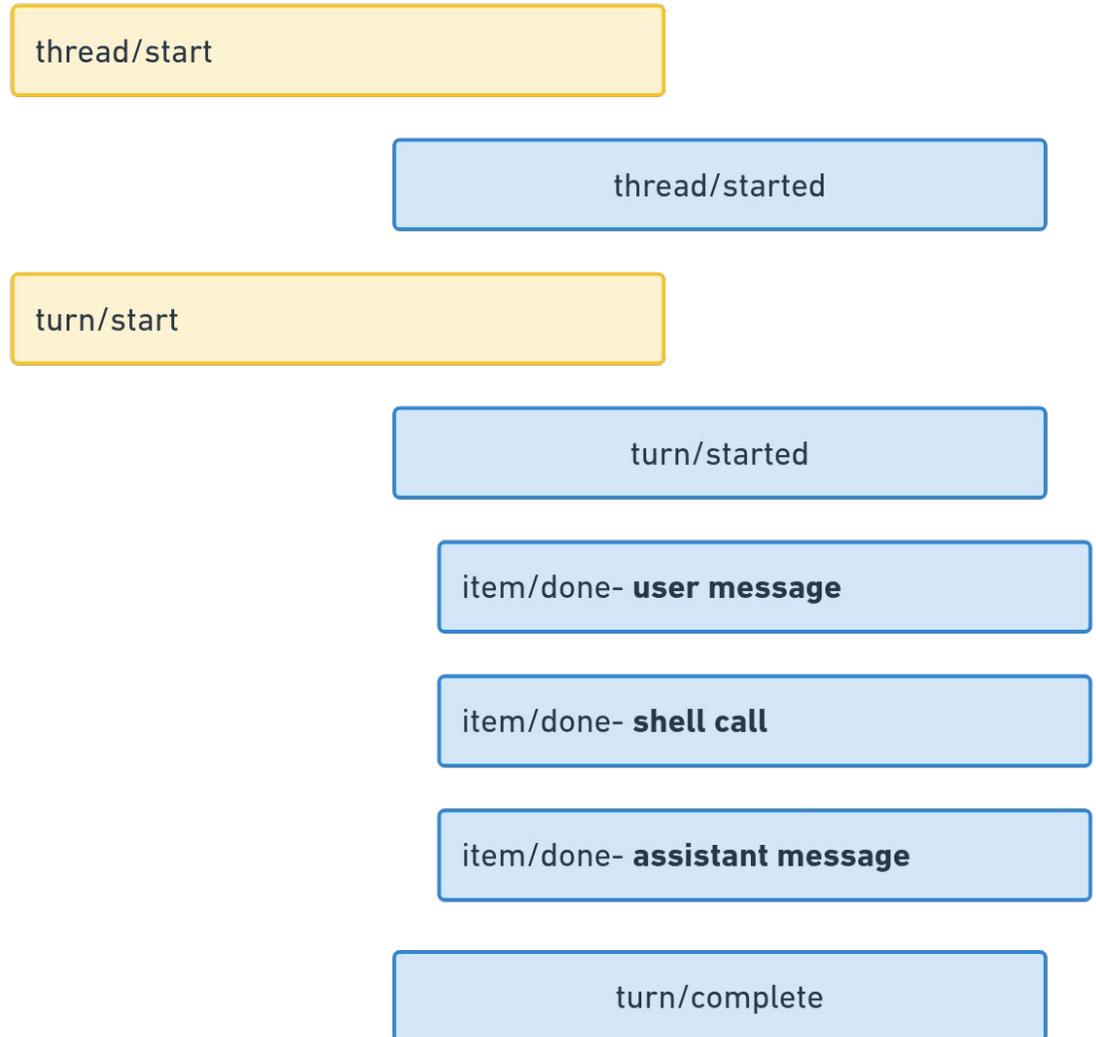
- It's a loop!
- Devil is in the details
  - Tool set
  - Sandbox
  - Truncation
  - Compaction
  - Environment information
  - Subagents
  - AGENTS.md/Rules
- Open-source
- Rust



# app-server

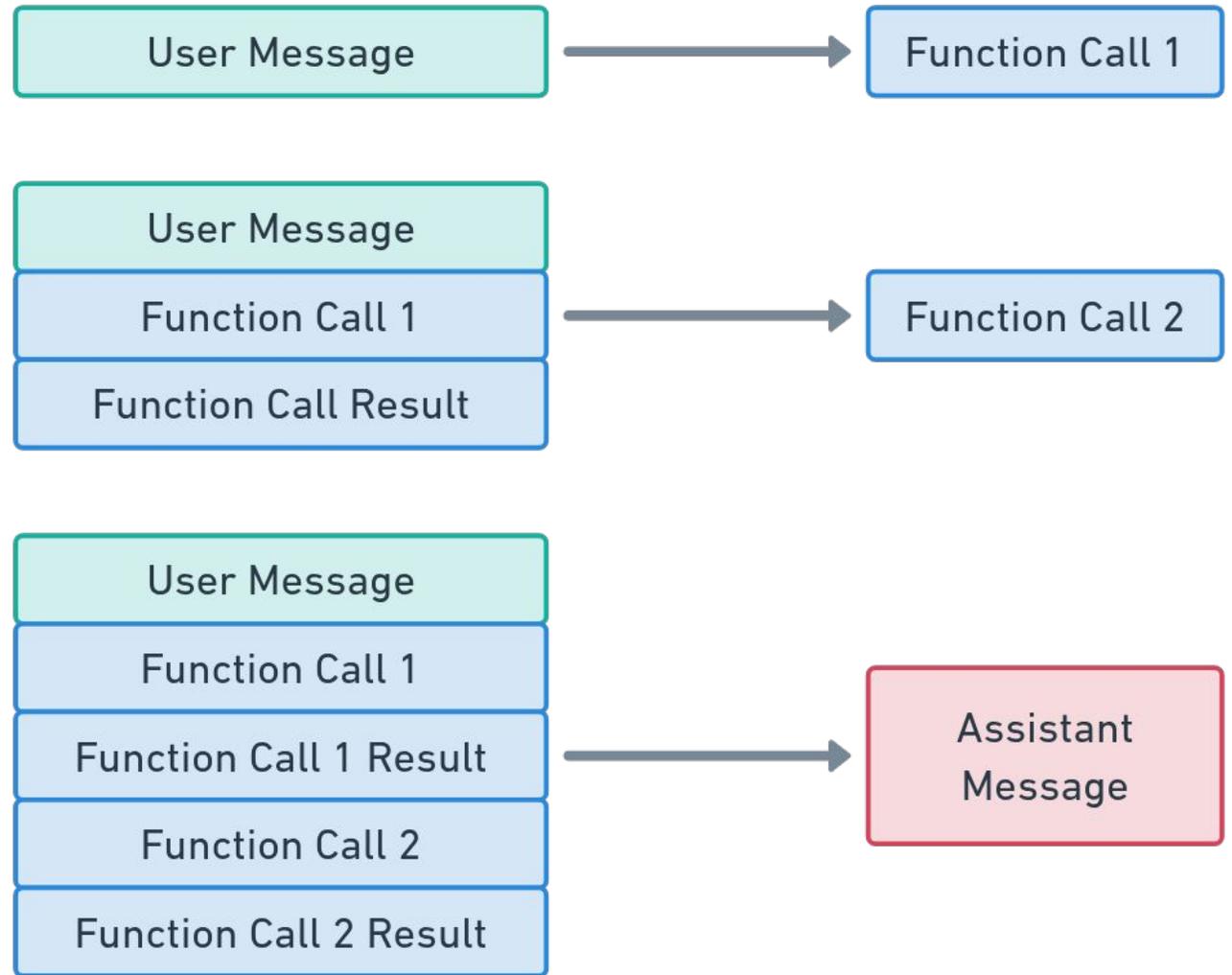
## API of the agent

- JSON-RPC
  - Can run on a different machine
- Main integration point
- Shared data types
  - Turn - user message to agent response
  - Thread
    - Item
      - Shell command
      - File Change
      - Todo list
      - Plan
      - Reasoning
      - Web-search



# Responses API

- Strongly-typed completions
- Items
  - message
  - function\_call
  - function\_call\_output
  - custom\_tool
  - custom\_tool\_output
- Compaction
- Web-search
- Image generation

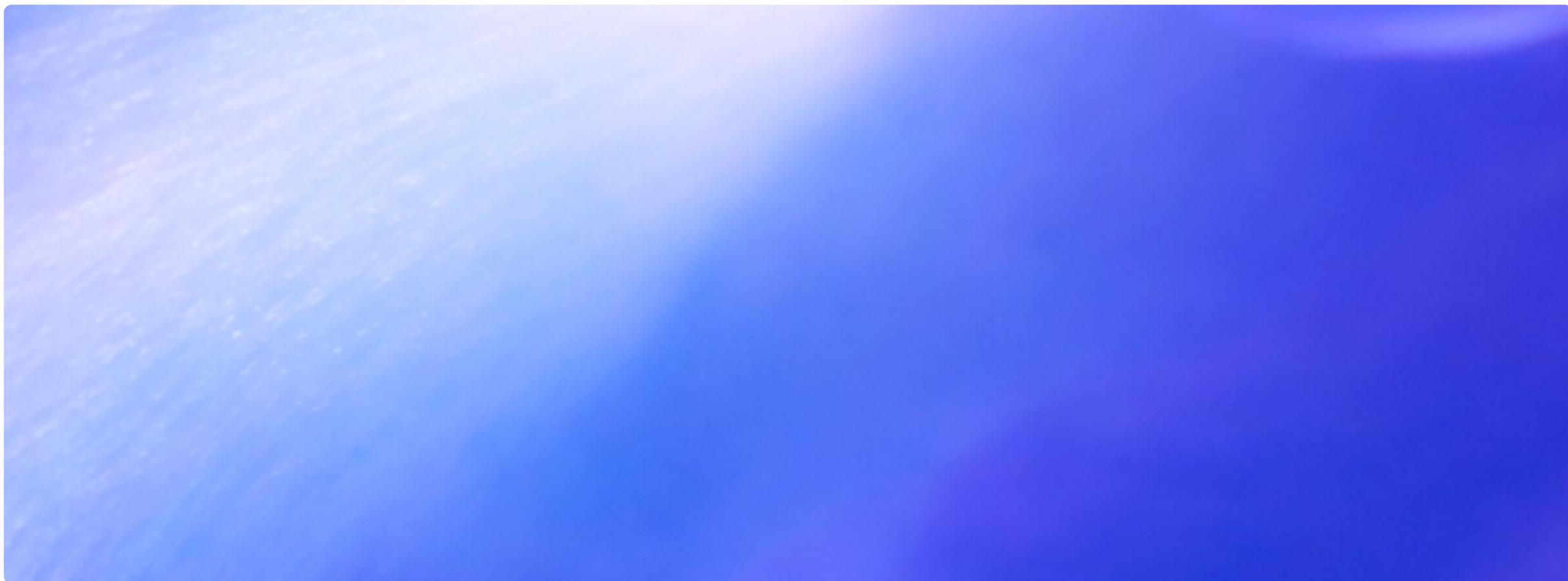


# Next Steps

- Test out advanced features and look out for /experimental features
- Have Codex run a long and complex task for you uninterrupted
- Leverage Codex SDK to build out additional use cases



# Q&A



# Resources

## Codex Documentation

*[developers.openai.com/codex](https://developers.openai.com/codex)*



## Codex Cookbooks

*[cookbook.openai.com/topic/codex](https://cookbook.openai.com/topic/codex)*



## Codex Changelog

*[developers.openai.com/codex/changelog](https://developers.openai.com/codex/changelog)*

