

OpenAI

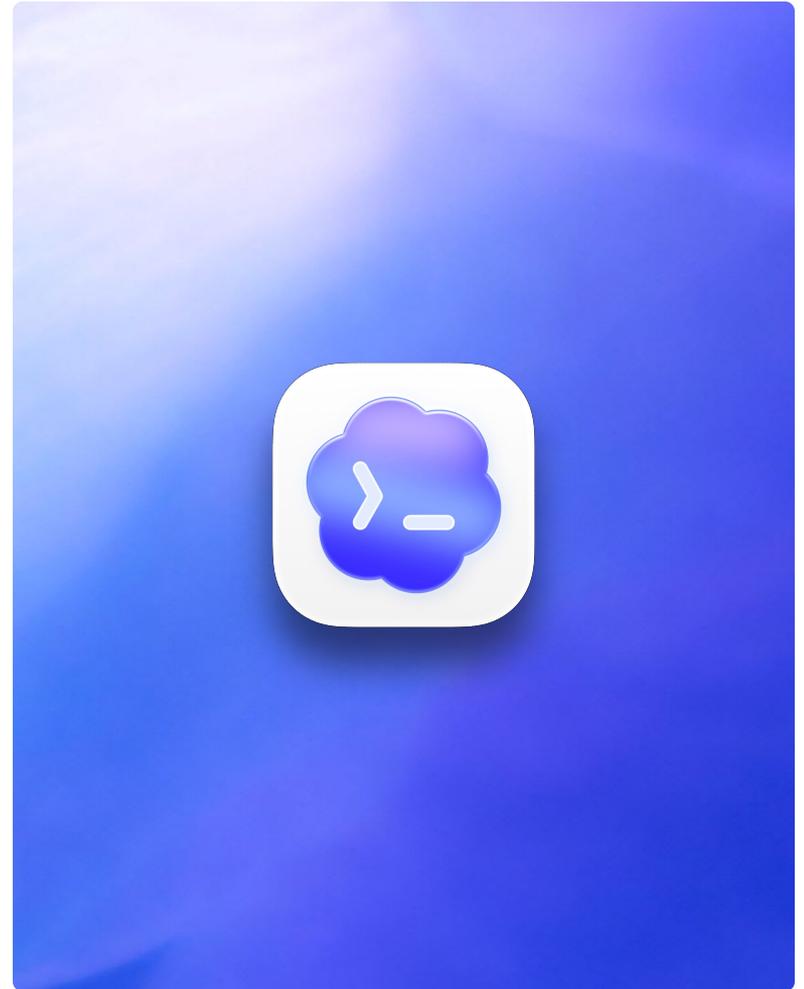
# Codex 101 - Introduction and Onboarding

Get up and running with  
OpenAI's coding agent



Name goes here

Title goes here



# Workshop Overview

## → Goals

- ◆ Get you up and running in Codex
- ◆ Demonstrate Success in an Agent-Native operating paradigm

## → Format

- ◆ Quick Concepts Overview
- ◆ Concept-Driven Lightning Talk Demos
- ◆ Skill building micro hackathon
- ◆ Q&A

→ You can use any of the Codex App, IDE, or CLI. The performance is the same, functionalities of clients differ slightly. We recommend the app



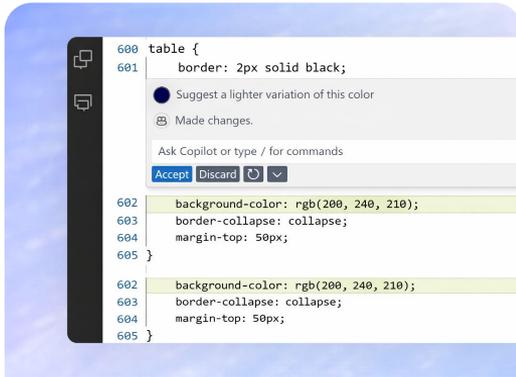
Merged

+217 -196

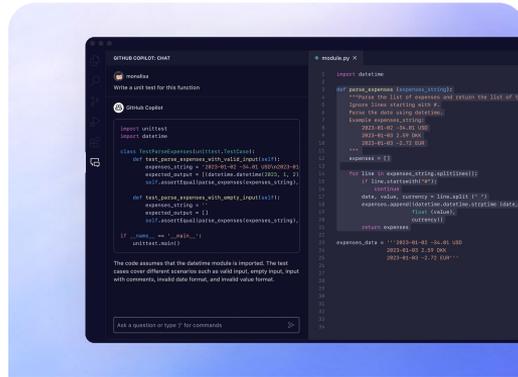
# Lightning Talks

- 01 Introduction and Setup
- 02 Understanding Agents
- 03 Using Codex in the SDLC
- 04 How to Delegate to Agents
- 05 How Teams are Using Codex

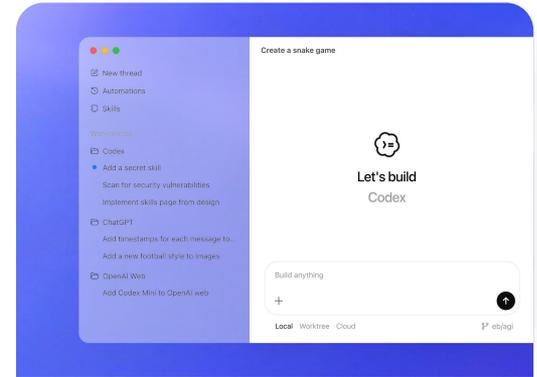
# Approaches to AI coding



Code  
complete



Pair  
programming



Agentic  
delegation

# One Agent, Everywhere you Work



## CLI

- Original & Popular UI
- Zero-Dependency Install
- Powerful Slash Commands (`/compact` `/fork` `/new`)
- Integration with ChatGPT Apps



## Codex App

- Easily run tasks in parallel across multiple workspaces
- Better/more flexible diff views
- Excellent visual surface for skills, worktrees & automations
- Great UI for workflows beyond writing code



## IDEs

(VS Code, JetBrains, Xcode)

- Fits naturally into IDE-based workflows
- Good for incremental tasks, quick assistance
- Great for 'chat-style' usage

# Installation & Auth



## CLI

```
1 npm install -g @openai/codex
2
3 # or
4
5 brew install --cask codex
```

Initiate login flow:

`codex login`

To start a session: `codex`



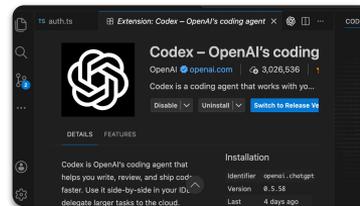
## Codex App

- [openai.com/codex/](https://openai.com/codex/)
- Download
- Drag into Applications
- Sign in with ChatGPT



## IDEs

- Open Extensions Pane
- Search “OpenAI Codex”
- Install
- Sign in with ChatGPT



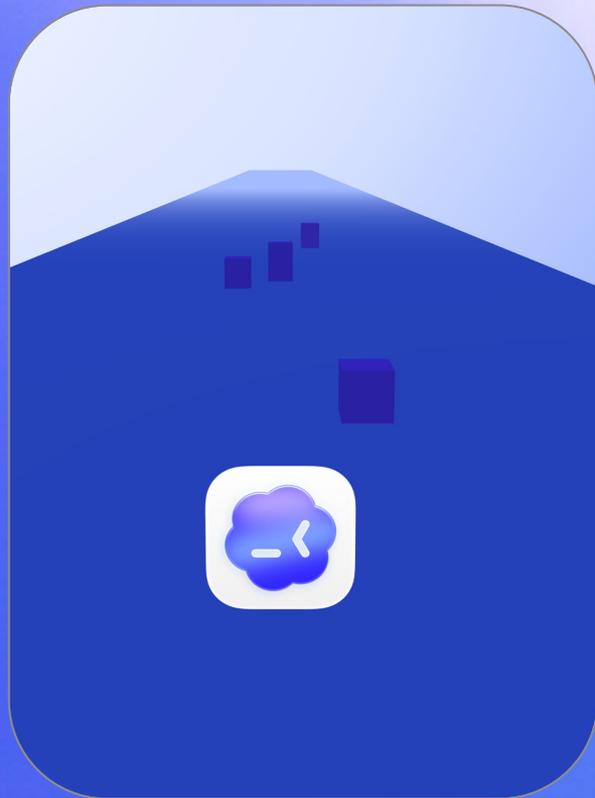
# Use Codex on one of your repos or an open source repo

## Need a repo? A few options

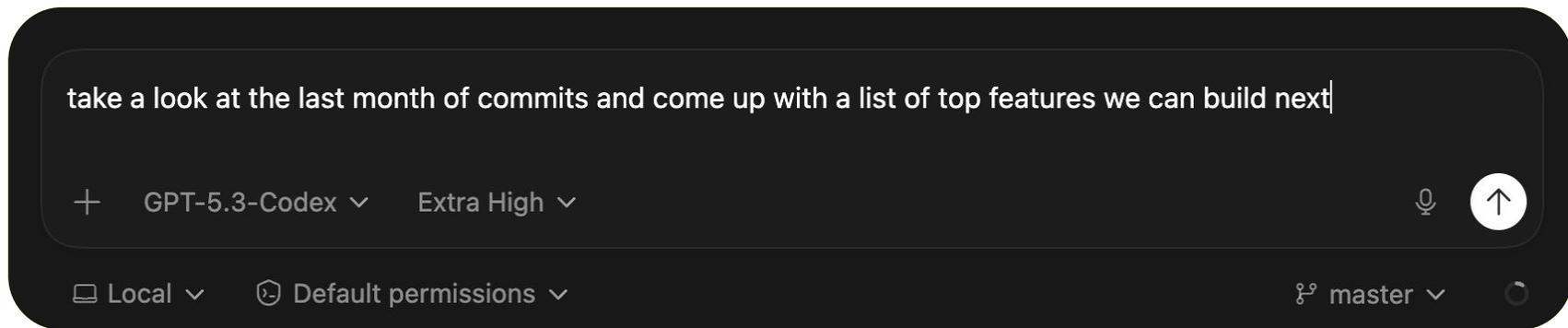
Link: [github.com/agentsmd/agents.md](https://github.com/agentsmd/agents.md)

Link:  
[github.com/greenleaf-oai/3js-follow-along-demo](https://github.com/greenleaf-oai/3js-follow-along-demo)

(Advanced) Link: [github.com/openai/codex](https://github.com/openai/codex)



# Send your first prompt to Codex



# Q&A

OpenAI



OpenAI

# Understanding Agents

What is the Agent Loop?



# Codex Agentic Loop

## It starts with a prompt

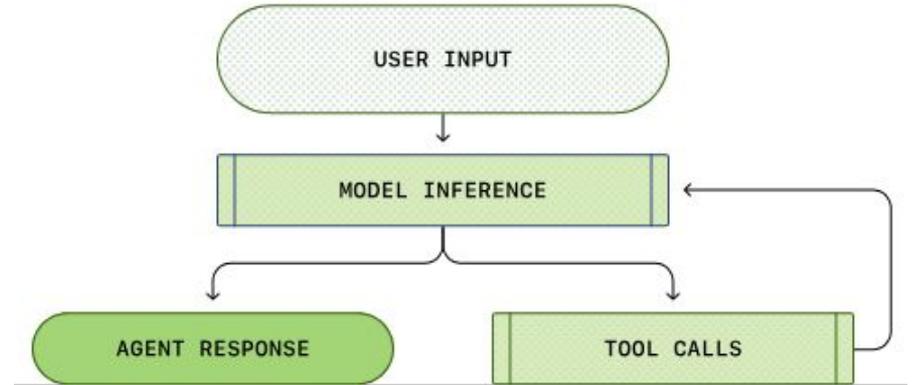
Give the agent initial direction which files to read and what your goals are

## Agent calls Tools

Reads your code, Searches company knowledge, Writes to a file, *loops* until task completion

## Builds the conversation history

Every action taken & every file read builds the models *context*



# What Is Context?

## Why does it matter?

### The Agent's Working Memory

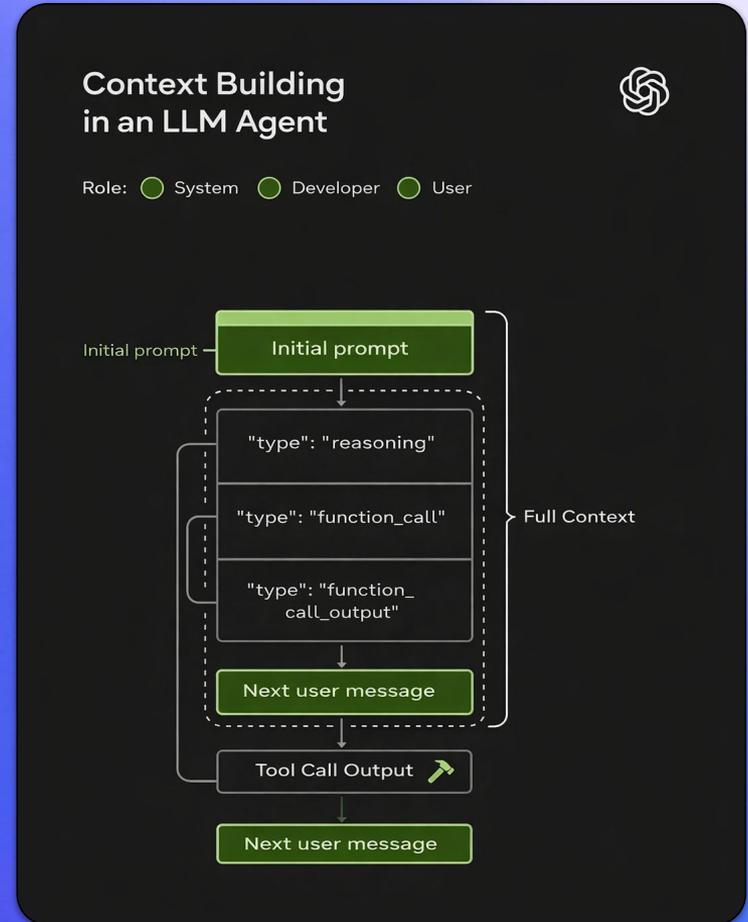
Context is the total information an agent has to complete its task

### The Context Window

Codex can hold hundreds of documents in context at once (400-600 pages of text)

### Why it Matters

Focused context leads to safer, more accurate, and more consistent agent results



# Codex Manages Context For You

- Intelligent Context Compression
- Auto compaction runs in the turn loop when token usage crosses a model-specific threshold ~90%

I'll now zoom into listener-attach and request/response wiring sections in `CodexMessageProcessor` since those are often prone to lifecycle issues.

Explored 1 file, 3 searches

I identified several likely review targets; I'll quickly inspect approval/task cleanup and state handling files to confirm whether any of these map-based state can leak on cancellation or shutdown.

Explored 1 file, 1 search

Context automatically compacted

Worked for 4m 3s

Pulled the latest `main` and did a broad review of this codebase's major areas (especially the `app-server`, `core`, and related message/turn handling flow), then focused on the largest runtime path ( `initialize` → `message processing` → `thread/turn execution` → `outgoing callbacks` ).

1. [P1] Hard panic on unreachable-but-invalid protocol path in `CodexMessageProcessor`

- File: `codex_message_processor.rs` (lines 488-490)
- `process_request` panics on `ClientRequest::Initialize` with "Initialize should be handled in `MessageProcessor`".

# Q&A

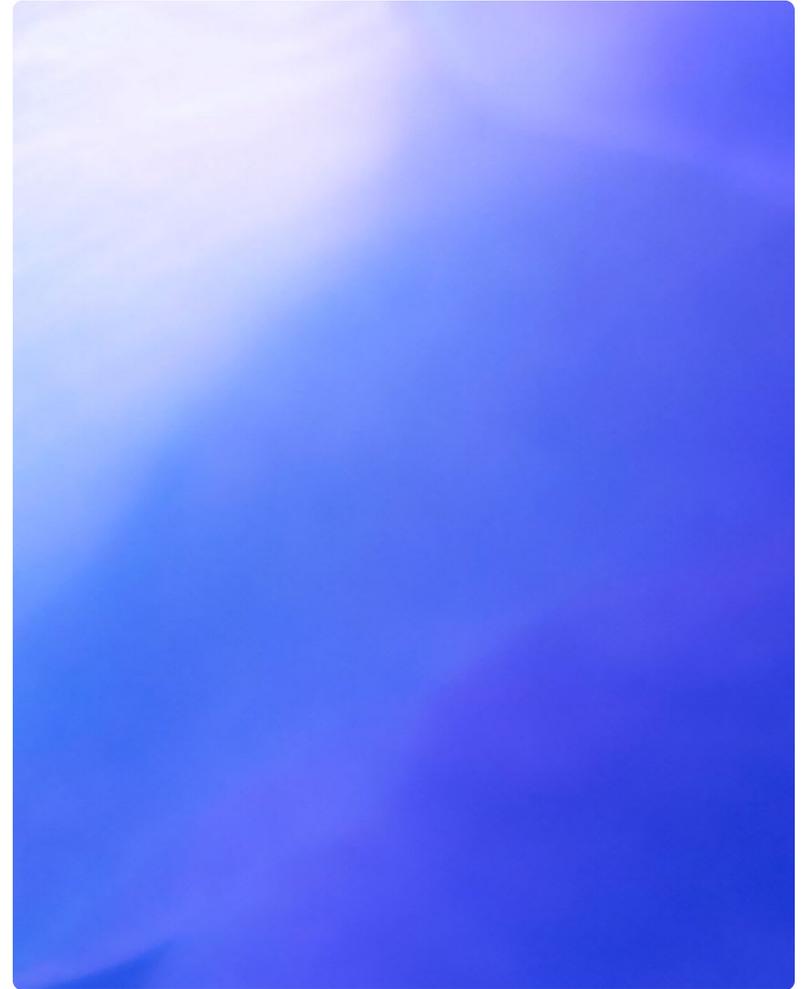
OpenAI



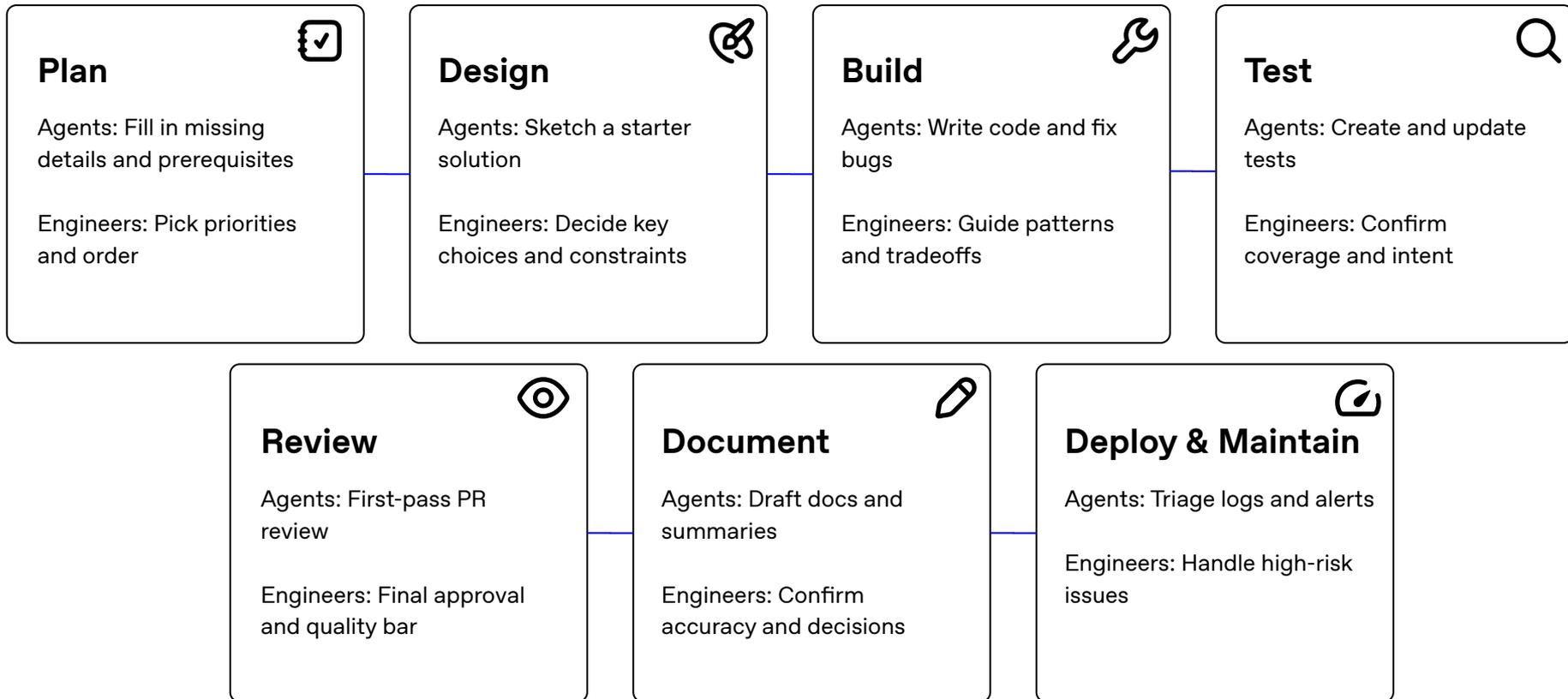
OpenAI

# Using Codex in the SDLC

Ways to leverage Codex in your work

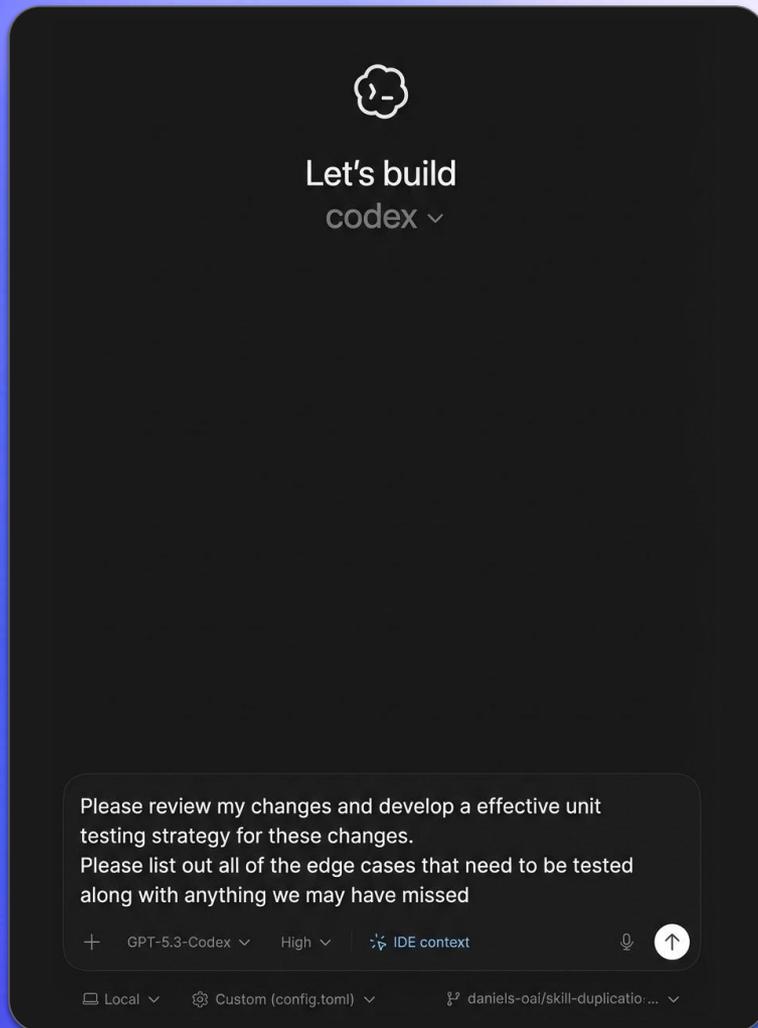


# Codex turbocharges engineers across the entire software lifecycle



# Time to build together

- Send a read-only prompt with @ mention
- Build a simple feature (+ run tests)
- Build a harder feature using “plan mode”
- Add image context
- Review code before deploying
- Document changes



**Delegate** to codex, and **free yourself** to think more deeply about architecture and design



# Q&A

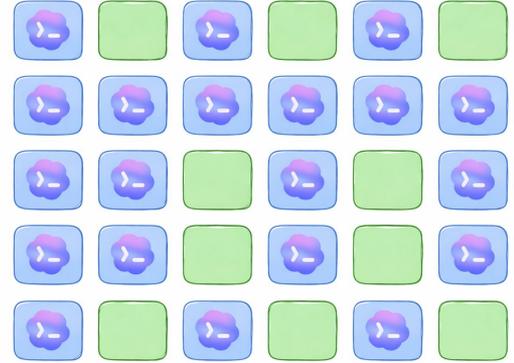
OpenAI



OpenAI

# How To Effectively Delegate to Agents

Become a power user fast



Codex wrote **85%**  
of the projects' code

# What power users know

- 01 Instead of a perfect prompt, build perfect context
- 02 Faster validation means better outcomes
- 03 Use the agent to learn, not vibe code

# Mini Codex prompt template

Four fields that keep work scoped,  
reviewable, and repeatable



## Goal

The outcome you want, what you are changing, and why it matters



## Context Pointers

Relevant files or any examples of desired behavior or existing patterns



## Constraints

Do and do not rules, conventions, and style expectations



## Done When

Tests and checks to pass, commands to run, and expected behavior verified

# Prompting: Why it matters + how to do it well

A good prompt sets scope, constraints, and success criteria so Codex can execute predictably

**Keep it minimal: goal + constraints + “done”:** Codex performs best with a clear target and a finish line, not a long narrative. Tell Codex what not to do when needed

**Bake in verification:** Specify how you will validate (tests, lint, build). If it cannot run them, it should tell you exactly what to run and what to look for

**Provide clear code pointers with @ mention:**  
Reference the relevant files in the codebase to keep edits scoped and accurate

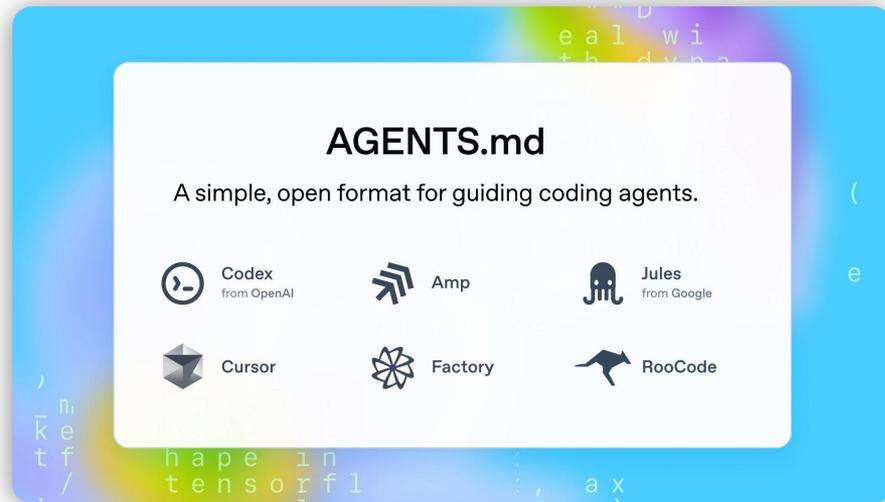
**Paste the full stack trace when debugging:** Include the error, repro steps, and environment details so Codex can pinpoint root cause faster

**Use open ended prompts intentionally:** Ask for options, tradeoffs, risks, and recommended approaches when you want guidance, not implementation

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

# Adding AGENTS.md helps make Codex smarter

- Codex starts each run with limited project knowledge
- AGENTS.md makes your conventions and workflows load automatically
- Result: more consistent edits, fewer retries, safer changes
- Use `/init` to generate a starter AGENTS.md



# How Codex discovers guidance

Discovery follows this precedence order



## 🌐 Global (~/.codex/AGENTS.md)

- Codex uses only the first non-empty file at the home directory level
- Keep concise and use for personal defaults

---

## 📁 Project (AGENTS.md)

### 📁 Directory (AGENTS.md)

- Starts at the project root (usually Git root) and walks down to the current working directory
- Concatenates files from the root down
- Files closer to current directory override earlier guidance for specific context and guidance

# What AGENTS.md typically includes

## Common sections:

- Repo map: key directories, important modules
- How to run the project: install, build, test, lint
- Engineering guardrails: conventions, PR expectations, “do not” rules, security constraints
- How work gets verified: what “done” means and how to validate it

Tip: You can always ask Codex to do a retrospective and update the Agents.md for you to avoid future mistakes or get to an end result faster

```
# Sample AGENTS.md file
```

```
## Dev environment tips
```

- Use ``pnpm dlx turbo run where <project_name>`` to jump to a package instead
- Run ``pnpm install --filter <project_name>`` to add the package to your worksp
- Use ``pnpm create vite@latest <project_name> -- --template react-ts`` to spin
- Check the name field inside each package's package.json to confirm the right

```
## Testing instructions
```

- Find the CI plan in the .github/workflows folder.
- Run ``pnpm turbo run test --filter <project_name>`` to run every check defin
- From the package root you can just call ``pnpm test``. The commit should pass
- To focus on one step, add the Vitest pattern: ``pnpm vitest run -t "<test na``
- Fix any test or type errors until the whole suite is green.
- After moving files or changing imports, run ``pnpm lint --filter <project_na``
- Add or update tests for the code you change, even if nobody asked.

```
## PR instructions
```

- Title format: [`<project_name>`] `<Title>`
- Always run ``pnpm lint`` and ``pnpm test`` before committing.

# AGENTS.md best practices

## 🔍 Keep it brief and actionable

Commands, paths, rules, and pitfalls. Skip generic explanations. Most of OpenAI's AGENTS.md files are less than 100 lines

## 📁 Point to task-specific .md files

Reference supporting markdown (.md) files in main [Agents.md](#) for specific workflows and tasks:

- Plans.md for planning general large tasks
- Code\_review.md for code review instructions
- Architecture.md for changes that require architecture modifications

# Q&A

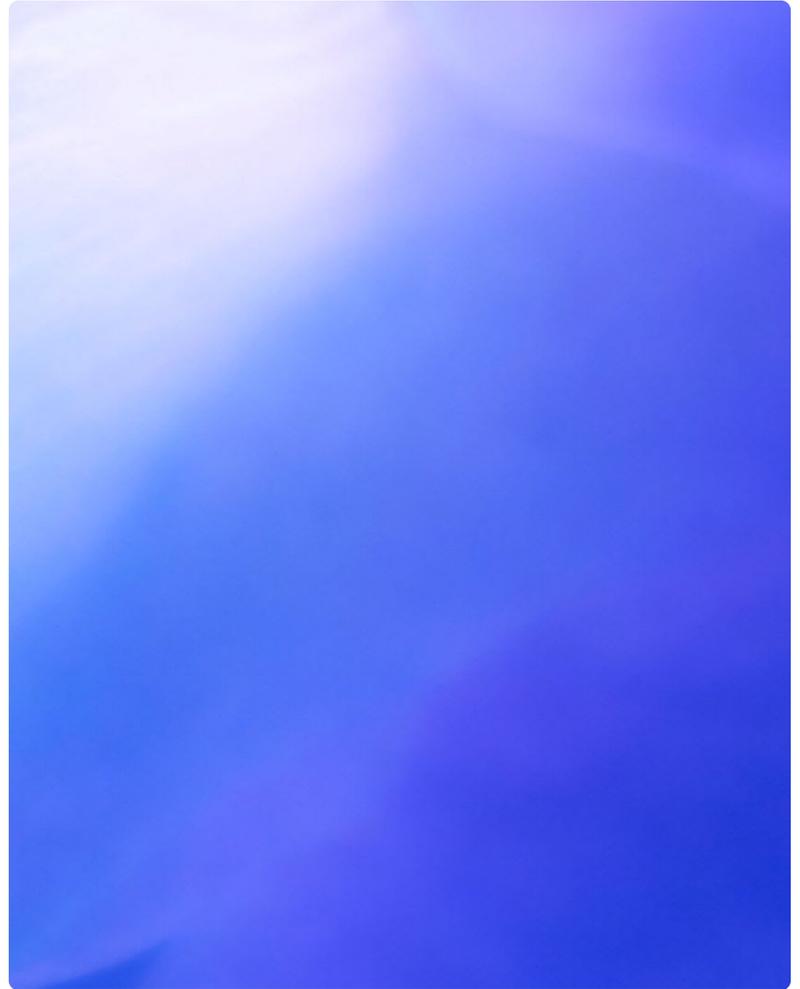
OpenAI



OpenAI

# How Teams are Using Codex

Understand the art of the  
possible



# Codex is deeply integrated with team workflows

```
components/theme-provider.tsx
Comment on lines +17 to +30

17 + useEffect(() => {
18 +   const stored = localStorage.getItem("theme") as Theme | null
19 +   if (stored) {
20 +     setTheme(stored)
21 +     document.documentElement.classList.toggle("dark", stored === "dark")
22 +   } else {
23 +     const prefersDark = window.matchMedia("(prefers-color-scheme: dark)").matches
24 +     setTheme(prefersDark ? "dark" : "Light")
25 +     document.documentElement.classList.toggle("dark", prefersDark)
26 +   }
27 + }, [])
28 +
29 + useEffect(() => {
30 +   document.documentElement.classList.toggle("dark", theme === "dark")

```

chatgpt-codex-connector bot on Aug 29

[P1] Prevent theme persistence effect from undoing saved dark mode

The second `useEffect` persists the current theme to `localStorage` and toggles the `dark` class on every render. On the initial mount the state defaults to `"Light"`, so this effect runs once before the first effect has read the stored preference, which removes the `dark` class and writes `"Light"` to `localStorage`. Users who previously selected dark mode (or prefer dark mode) will see a brief flash to the light theme before the subsequent re-render restores dark mode, defeating the prelad script's purpose. Consider skipping the persistence effect until the initial theme has been resolved (e.g. track an `initialized` flag or run both operations in a single effect).

Useful? React with 👍/👎.

Greg Brockman Today at 12:31

I've set `combo.raise_all_timeouts=True`, but i still get this spew (and eventually a timeout) while waiting in a down why, but also lmk if there's just some flag i should add:

```
still waiting for /root/code/openai/lib/strawberry/strawberry/backend/worker_group/run_rpc.py:67 - [strawberry.pretraining.train_loop._handle_step_output]: waiting for step output from the step: 1
```

33 replies

Han Lu Today at 12:34

Most of the timeouts are managed by strawberry, such as the one you are looking at for the train loop. Please `!strawberry.presets.combo:RAISE_ALL_TIMEOUTS` to the command.

Steve Lee Today at 12:34

i think we need to update the presets with new timeouts too

@Codex can you take a look?

👍 1 🗨

Codex APP Today at 12:39

Summary

- Extended the `!strawberry.presets.combo:RAISE_ALL_TIMEOUTS` preset to raise the new per-step timeout knobs so the combo shortcut still covers every train-loop timeout.
- Expanded unit tests for the combo preset and legacy flag passthrough to assert the added timeout fields.

Testing

- 🟢 `SKIP_TOR...`

See more

View task

```
name: Codex Auto-Fix CI on Failure

on:
  workflow_run:
    workflows: ["CI"]
    types: [completed]

jobs:
  auto-fix:
    - name: Run Codex
      uses: openai/codex-action@main
      id: run-codex
      with:
        openai_api_key: ${{ secrets.OPENAI_API_KEY }}
        prompt: >-
          Fix all failing tests.
          - name: Create pull request with fixes
            with:
              commit-message: "Auto-fix failing tests via Codex"
              base: ${{ env.FAILED_HEAD_BRANCH }}
              title: "Auto-fix failing CI via Codex"
```

## Code Review

Auto and reactive code reviews to help accelerate the review process and prevent issues in production

## Slack and Linear Mentions

Tag `@Codex` in a Slack channel or Linear issue and it will gather context, choose the right environment, and answer with a link to the completed task

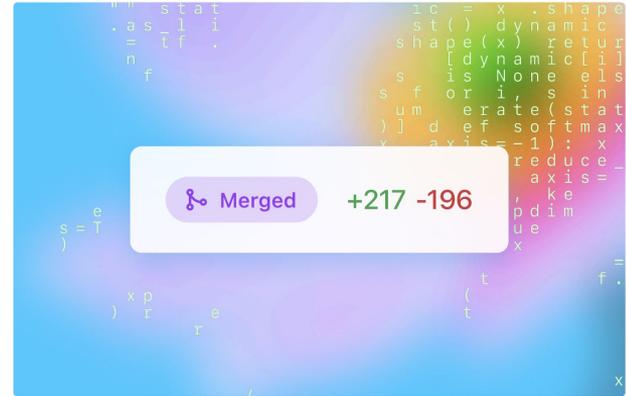
## Codex SDK

Build your own integrations with the Codex SDK. Auto-fix CI failures or kick off root-cause analysis when an incident is declared

# What we've seen with OpenAI's Engineers

- Some teams are entirely “hands off” - they prompt everything and have Codex write all code
- Codex reviews 100% of PRs internally
- Agents work proactively in the background
- Early versions of 5.3-Codex debugged its own training runs

## How OpenAI uses Codex



# Cisco is transforming its engineering org with Codex



The biggest gains came when we stopped thinking about Codex as a tool and started treating it as part of the team.”

Ryan Brady  
Principal Engineer

1,500

Engineering hours saved per month

10–15x

Faster defect remediation

20%

Reduction in build times

# Datadog ships more reliably with Codex Code Review



Codex changed my mind for what code review should be. It's not about replicating our best human reviewers. It's about finding critical flaws and edge cases that humans struggle to see when reviewing changes in isolation."

Brad Carter  
Engineering Manager at Datadog

22%

of past incidents had risks  
Codex flagged

1,000+

DataDog engineers  
using Codex weekly

# Rakuten fixes issues twice as fast with Codex



We don't just care about generating code quickly. We care about shipping safely. Speed without safety is not success. Our role is not to check every line of code anymore. Our role is to define clearly what we want and establish how to verify it.

Yusuke Kaji  
General Manager of AI at Rakuten

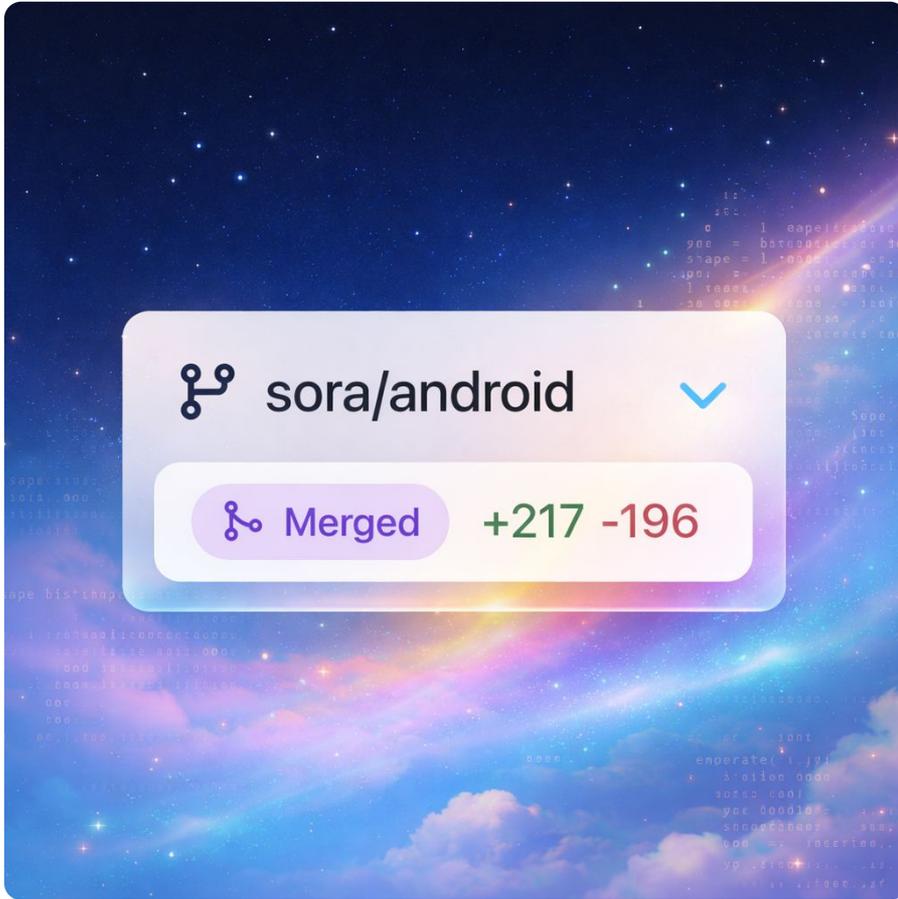


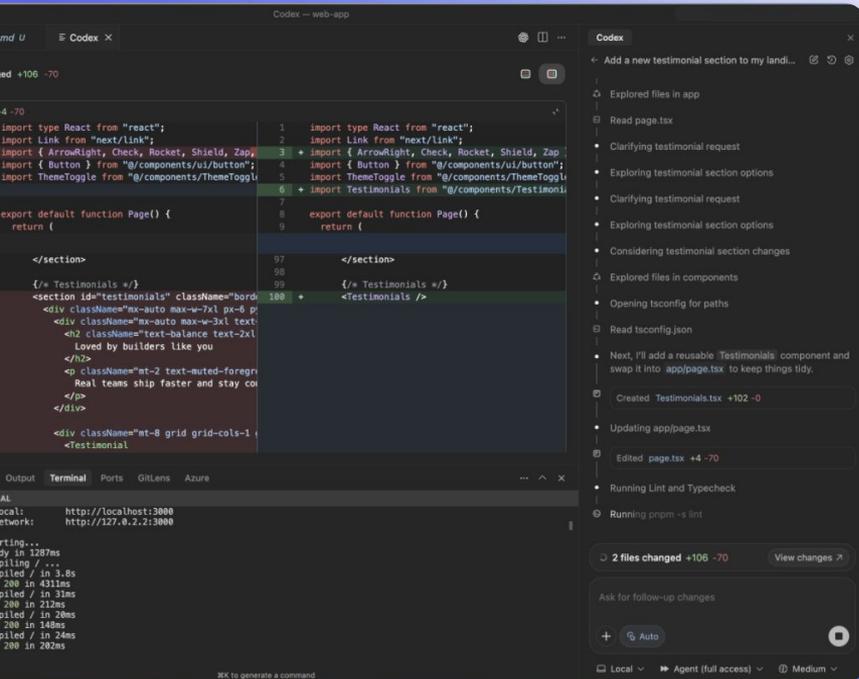
# 50%

Reduction in Mean Time to  
Repair (MTTR)

# Shipping Sora in 28 days with Codex

- Lean team of 4 engineers
- 18-day to prototype and 10 days to launch
- Codex wrote ~**85%** of the code, humans focused on architecture and review
- Parallel Codex workstreams: debug, refactor, tests, reliability
- **99.9%** crash-free, 4.9/5, #1 on Google Play





I'll leave everyone with 3 next steps:

- 1) Use Codex in a real repo
- 2) Set up Agents.md
- 3) Give it real tasks

We have additional sessions scheduled to dive deeper!

# 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)*

