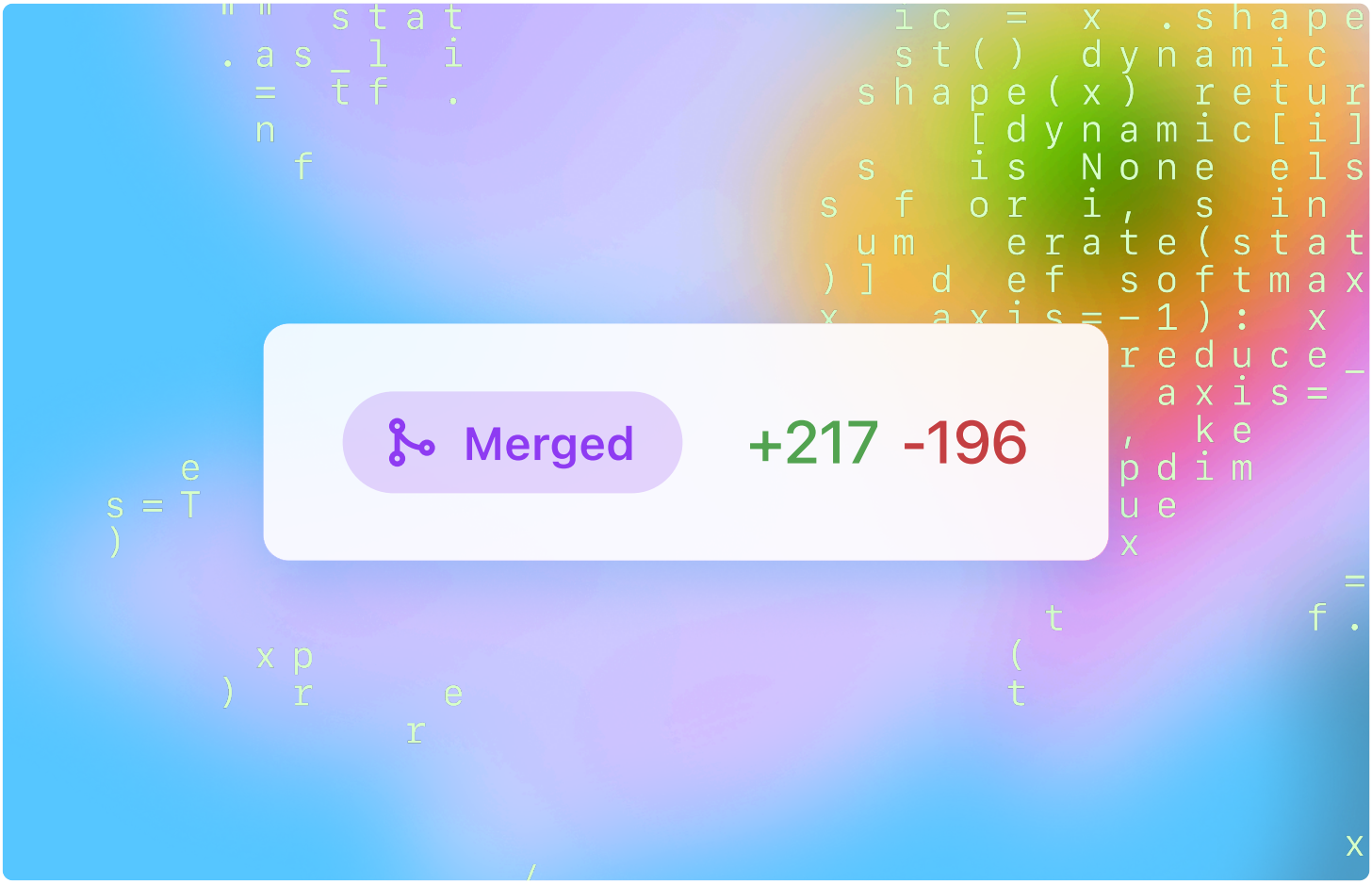
OpenAI

How OpenAI uses Codex



Contents

Introduction

Codex is used daily across numerous technical teams at OpenAI like Security, Product Engineering, Frontend, API, Infrastructure, and Performance Engineering. Teams are using it to accelerate a range of engineering tasks, from understanding complex systems and refactoring large codebases to shipping new features and resolving incidents under tight deadlines.

Drawing from interviews with OpenAI engineers and internal usage data, we've compiled use cases and best practices that highlight how Codex helps our teams move faster, improve work quality, and manage complexity at scale.

Code understanding

Codex helps our teams get up to speed quickly in unfamiliar parts of the codebase when onboarding, debugging, or investigating an incident.

They often use Codex to locate the core logic of a feature, map out relationships between services or modules, and trace data flow through a system. It also helps surface architecture patterns or missing pieces of documentation that would otherwise require significant manual effort to generate.

During incident response, Codex helps engineers ramp into new areas quickly by surfacing interactions between components or tracing how failure states propagate across systems.

Anecdotes from our teams

> *When I fix a bug, I use Ask mode to see where else in the codebase the same issue might appear.*
>
> **Performance Engineer, Retrieval Systems**

> *When I'm on-call, I paste the stack trace and ask Codex where the auth flow lives. It jumps straight to the right files so I can triage fast.*
>
> **Site Reliability Engineer, API Platform**

> *Codex answers my 'Where would I do this?' repo questions across Terraform and Python way faster than grep.*
>
> **DevOps Engineer, Infrastructure Services**

Try using Codex for code understanding with these sample prompts:

- [ ] Where is the authentication logic implemented in this repo?
- [ ] Summarize how requests flow through this service from entrypoint to response.
- [ ] Which modules interact with [insert module name] and how are failures handled?

Refactoring and migrations

Codex is commonly used to make changes that span multiple files or packages. For example, when engineers are updating an API, changing how a pattern is implemented, or migrating to a new dependency, Codex makes it easy to apply changes consistently.

It's especially useful when the same update needs to be made across dozens of files, or when the update requires awareness of structure and dependencies that aren't easily caught with a regex or find-and-replace.

They're also using it for code cleanup by breaking up oversized modules, replacing old patterns with modern ones, or preparing code for better testability.

Anecdotes from our teams

Codex swapped every legacy `getUserById()` *for our new service pattern and opened the PR. It did in minutes what would've taken hours.*

Backend Engineer, ChatGPT Web

To clear launch blockers, I have Codex scan for every instance of the old pattern, summarize the impact in Markdown, then open PRs with the fixes.

Product Engineer, ChatGPT Enterprise

Try using Codex for refactoring and migrations with these sample prompts:

- [ ] Split this file into separate modules by concern and generate tests for each one.
- [ ] Convert all callback-based database access to async/await.

Performance optimization

Codex is used to identify and address performance bottlenecks.

During tuning or reliability efforts, engineers prompt Codex to analyze slow or memory-intensive code paths, such as inefficient loops, redundant operations, or costly queries and suggest optimized alternatives, often resulting in meaningful gains in efficiency and reliability.

Codex is also used to support code health by identifying risky or deprecated patterns that are still in active use. Our teams lean on it to help reduce long-term tech debt and proactively prevent regressions.

Anecdotes from our teams

> *I use Codex to scan for repeated expensive DB calls. It's great at flagging hot paths and drafting batched queries I can later tune.*
>
> **Infrastructure Engineer, API Reliability**

> *Codex is great for spotting performance issues quickly— I save 30 minutes of work by spending 5 minutes on a prompt.*
>
> **Platform Engineer, Model Serving**

Try using Codex for performance optimization with these sample prompts:

- [ ] Optimize this loop for memory efficiency and explain why your version is faster.
- [ ] Find repeated expensive operations in this request handler and suggest caching opportunities.
- [ ] Suggest a faster way to batch DB queries in this function.

Improving test coverage

Codex helps engineers write tests faster — especially in places where coverage is thin or completely missing.

When working on a bug fix or refactor, engineers often ask Codex to suggest tests that cover edge cases or likely failure paths. For new code, it can generate unit or integration tests based on the function signature and surrounding logic.

Codex is particularly helpful for identifying boundary conditions like empty inputs, max length, or unusual but valid states that are often missed in initial tests.

Anecdotes from our teams

> *I point Codex at low-coverage modules overnight and wake up to runnable unit-test PRs.*
>
> **Frontend Engineer, ChatGPT Desktop**

> *When switching mono-repo branches is painful, I have Codex write the tests and kick-off CI while I keep working on my branch.*
>
> **Backend Engineer, Payments & Billing**

Try using Codex for improving test coverage with these sample prompts:

- [ ] Write unit tests for this function, including edge cases and failure paths.
- [ ] Generate a property-based test for this sorting utility.
- [ ] Extend this test file to cover missing scenarios around null inputs and invalid states.

Increasing development velocity

Codex helps teams move faster by accelerating both the start and end of the development cycle.

When kicking off a new feature, engineers use it to scaffold boilerplate — generating folders, modules, and API stubs to get runnable code up quickly without hand-wiring every piece.

As projects approach release, Codex helps meet tight deadlines by handling smaller but essential tasks like triaging bugs, filling in last-mile implementation gaps, and generating rollout scripts, telemetry hooks, or config files.

It's also used to turn product feedback into starter code. Engineers often paste in a user request or spec and have Codex generate a rough draft they can return to and refine later.

Anecdotes from our teams

I was in meetings all day and still merged 4 PRs because Codex was working in the background.

Product Engineer,
ChatGPT Enterprise

Codex helped ship 3-4 low-priority fixes perfectly that would've languished in the backlog, which was super empowering.

Full-Stack Engineer,
Internal Tools

Try using Codex for increasing development velocity with these sample prompts:

- [ ] Scaffold a new API route for POST /events with basic validation and logging.
- [ ] Generate a telemetry hook for tracking success/failure of the new onboarding flow, using this template [insert example of your telemetry code].
- [ ] Create a stub implementation based on this spec: [insert spec or product feedback].

Staying in flow

Codex helps our engineers stay productive when their schedules are fragmented and filled with interruptions.

It's used to capture unfinished work, turn notes into working prototypes, or spin off exploratory tasks that can be revisited later. This makes it easier to pause and resume work without losing context, especially when they're on call or have a lot of meetings.

Anecdotes from our teams

If I spot a drive-by fix, I fire a Codex task instead of swapping branches and review its PR when I'm free.

Backend Engineer,
ChatGPT API

I routinely forward Slack threads, Datadog traces, issues and more to Codex so I can stay focused on high priority work.

API Engineer,
Infrastructure Observability

Try using Codex for staying in flow with these sample prompts:

- [ ] Generate a plan to refactor this service and split it into smaller modules.
- [ ] Stub out the retry logic and add a TODO — I'll fill in the backoff logic later.
- [ ] Summarize this file so I can pick up where I left off tomorrow.

Exploration and ideation

Codex is also useful for open-ended work like finding alternative solutions or validating design decisions. You can prompt for different ways of solving a problem, explore unfamiliar patterns, or pressure-test assumptions. This helps surface tradeoffs, expand design options, and sharpen implementation choices.

It's also used to identify related bugs. Given a known issue or deprecated method, Codex can identify similar patterns elsewhere in the code, making it easier to catch regressions or finish cleanup work.

Anecdotes from our teams

Codex helps me solve the cold-start problem — I paste a spec and docs and it scaffolds code or shows me what I forgot.

Product Engineer, ChatGPT Desktop

After I fix a bug I ask Codex where similar bugs might lurk, then spin follow-up tasks.

Performance Engineer, Retrieval Systems

Try using Codex for exploration and ideation with these sample prompts

- [ ] How would this work if the system were event-driven instead of request/response?
- [ ] Find all modules that manually build SQL strings instead of using our query builder.
- [ ] Rewrite this in a more functional style, avoid mutation and side effects.

Best practices

Codex works best when it's given structure, context, and room to iterate. Here are some of the habits OpenAI teams are cultivating to get consistent value out of it in day-to-day work.

Start with Ask Mode	For large changes, start by prompting Codex for an implementation plan using Ask mode, which then becomes the input for follow-up prompts when you switch to Code Mode. This two-step flow keeps Codex grounded and helps avoid errors in its output. Codex works best with well-scoped tasks that would take you or a teammate about an hour to complete or a few hundred lines of code to implement. As models improve, expect the size of the tasks it can take on to increase.
Iteratively improve Codex's development environment	Setting a startup script, environment variables, and internet access significantly reduces Codex's error rate. As you run tasks, look for build errors that can be corrected in Codex's environment configuration. This may take a few iterations, but gives significant efficiency gains in the long run.
Structure your prompt as if you are writing a Github Issue	Codex responds better when prompts mirror how you'd describe a change in a PR or issue. That means including file paths, component names, diffs, and doc snippets when relevant. Prompting with patterns like "Implement this the same way it's done in [module X]" improves results.
Use the Codex task queue as a lightweight backlog	Fire off tasks to capture tangential ideas, partial work, or incidental fixes. There's no pressure to generate a full PR in one go. Codex works well as a staging area you can return to when you're back in focus.

Use AGENTS.md to supply persistent context	Maintain an AGENTS.md file to help Codex operate more effectively in your repo across prompts. These files typically include naming conventions, business logic, known quirks, or dependencies Codex can't infer from the code alone. Learn more on structuring your AGENTS.md file in the docs.
Leverage "Best of N" to improve output	The Best-of-N feature lets you simultaneously generate multiple responses for a single task to quickly explore multiple solutions and pick the best one. For more complicated tasks, you can review several iterations and combine parts of different responses to get a stronger result.

Looking ahead

Codex is still in research preview, but it's already making a real impact in how we build, helping us move faster, write better code, and take on work that would've otherwise never been prioritized.

We're excited by the potential ahead — as our models get better and Codex becomes more deeply integrated into our workflows, we're looking forward to unlocking even more powerful ways to develop software with it. We'll continue to share what we learn along the way.

OpenAI