5 tips to turn OpenAI Codex into a more powerful AI Coding Agent

Discover 5 ways to make OpenAI Codex function like a true AI coding agent with Planning Mode, AGENTS.md, custom skills, shell tools, and a workflow for self-checking results.

The OpenAI Codex is more than just a tool for generating code snippets or handling minor tweaks. When set up correctly, the Codex can function much more like a real software engineer than many people realize — following instructions, understanding project context, using the terminal and workflow CLI effectively, handling changes across multiple files simultaneously, and even testing results before completing a task.

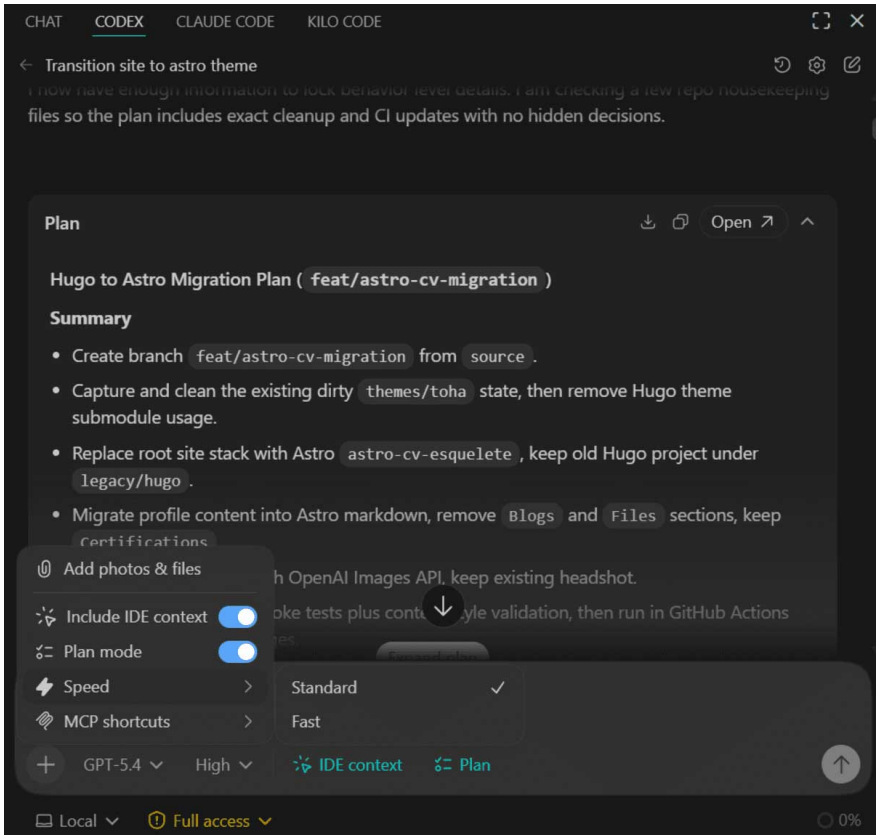
In this article, we'll explore five practical ways to make the Codex more useful for real-world programming tasks. Instead of viewing it as a simple code generation tool, the goal is to transform the Codex into an AI coding agent capable of handling lengthy tasks, closely following project workflows, and producing more reliable results.

Note that this is a personal opinion based on real-world experience, but the ideas in this article are not solely subjective. They are also based on official OpenAI documentation, recent research, and emerging trends in the 'vibe-coding' community.

1. Use Planning Mode for long and complex tasks.

According to OpenAI's guidelines, Planning Mode is particularly well-suited for complex tasks that are difficult to describe or involve multiple processing steps. This mode allows the Codex to gather more context, ask clarifying questions, and build a plan before beginning code modifications.

The major difference lies in the way it interacts. Instead of jumping straight into generating code, the Codex takes the time to understand the problem, check the codebase, and break the work down into more logical steps.

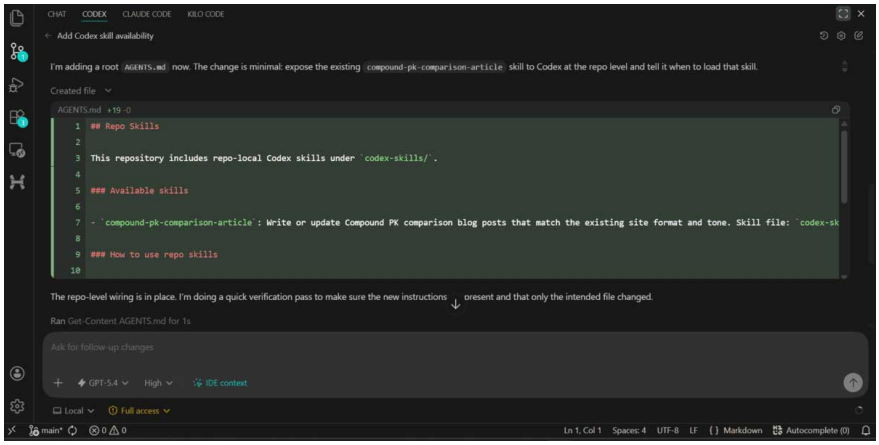


In practice, this helps the Codex handle lengthy workflows significantly more efficiently. The success of these types of tasks often lies not in writing a single piece of code, but in the ability to manage the workflow, constraints, checkpoints, and result verification process throughout the entire project.

2. Use AGENTS.md to manage project rules and 'memory'.

Many people think the **AGENTS.md** file is just a quick introduction to the Codex, but in reality, it's much more powerful.

According to OpenAI's documentation, Codex reads AGENTS.md files before starting work. Codex's CLI tool can even pre-generate a framework using the **/init** command , which users can then edit and commit for use in subsequent sessions.



The best thing about **AGENTS.md** is that it helps the Codex understand:

1. How does the project work?
2. What is the workflow like?
3. Which tools are allowed to be used?
4. Coding standards must be followed.

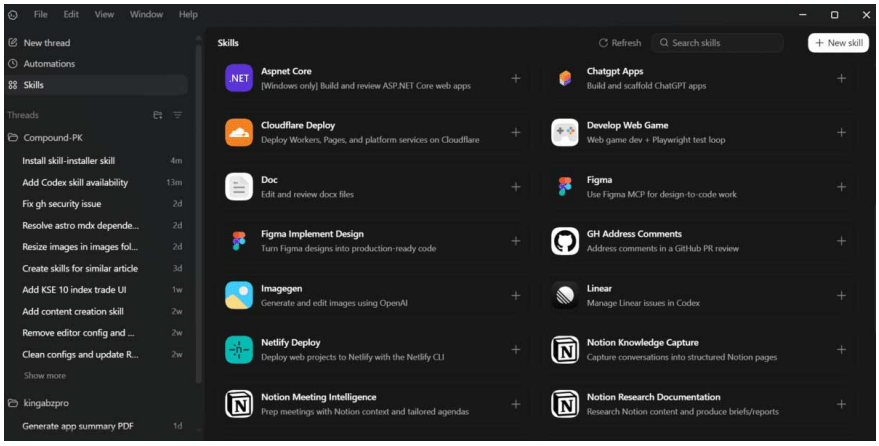
More importantly, it acts as a lightweight 'project memory' layer. This isn't the kind of personal memory like ChatGPT, but rather memory dedicated specifically to the project workflow.

OpenAI also recommends using persistent markdown files to store plans, execution instructions, and documentation in long-term workflows. Combined with the Codex's session resume capabilities, **AGENTS.md** becomes a highly effective way to maintain context across multiple work sessions.

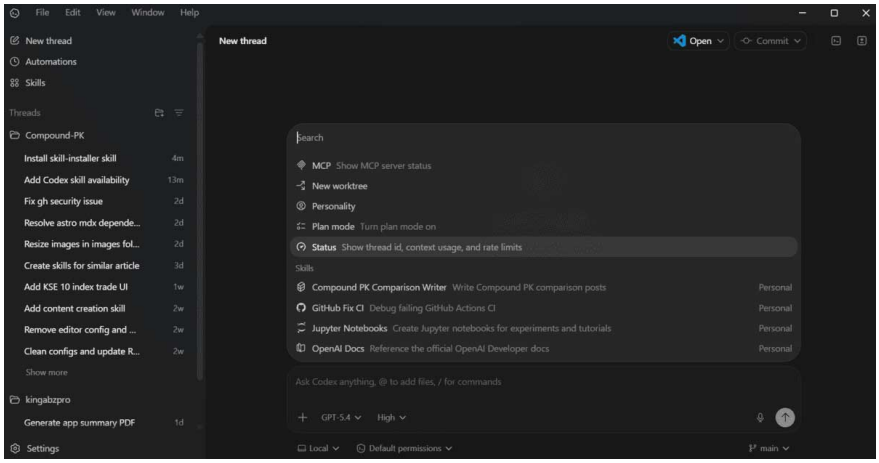
3. Create custom skills for repetitive workflows.

One of the most useful ways to extend the capabilities of the Codex is to use Skills.

According to OpenAI, skills are reusable packages that include tutorials, scripts, and accompanying resources, built around the **SKILL.md** file . They help standardize repetitive workflows or project-specific processes.



The Codex also includes several built-in system skills such as **$skill-creator** and **$skill-installer** , which help create and install new skills faster.



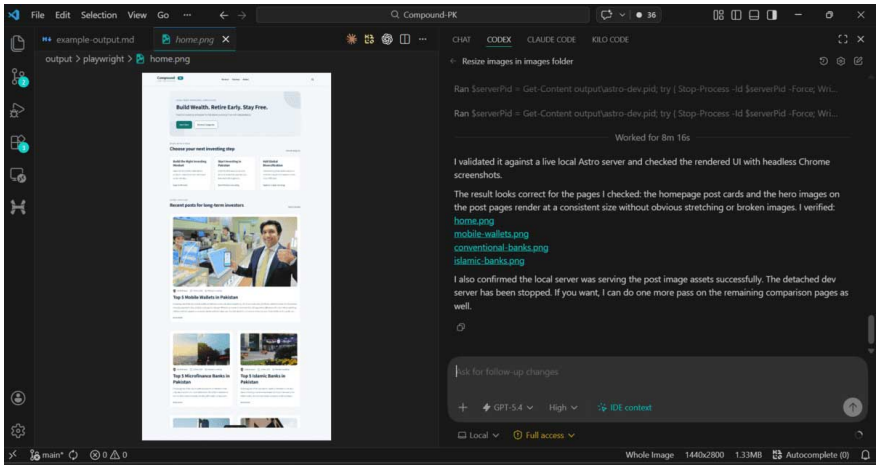
This is especially useful when your workflow isn't generic. Instead of relying entirely on AI's default behavior, you can create custom skills to teach the Codex how to handle project-specific tasks, use internal tools, interact with APIs, or automate publishing processes.

Skills can save a significant amount of time thanks to its ability to standardize formatting, handle CLI operations, and work with more reliable external services.

4. Enable the Codex to perform self-testing and verify the results.

This point becomes even more important with newer models like the GPT-5.4.

According to the official guidelines, the new model is better optimized for coding and multi-step workflows, especially in its verification loop, completion check, and tool usage in complex tasks. Simply put, the Codex is now less inclined to "just give the answer" the first time and is willing to continue testing until everything works correctly.



In practice, this allows the Codex:

1. write code,

2. run tests,
3. open the website or UI,
4. check the results,
5. fix errors,
6. Then, repeat the process until the task is completed as required.

To make the most of this capability, you should explicitly instruct the Codex to self-test its work. For example, tell it to run tests, open the application, review the UI, or verify the behavior on the website before completing the task.

5. Use shell tools to turn the Codex into a real coding agent.

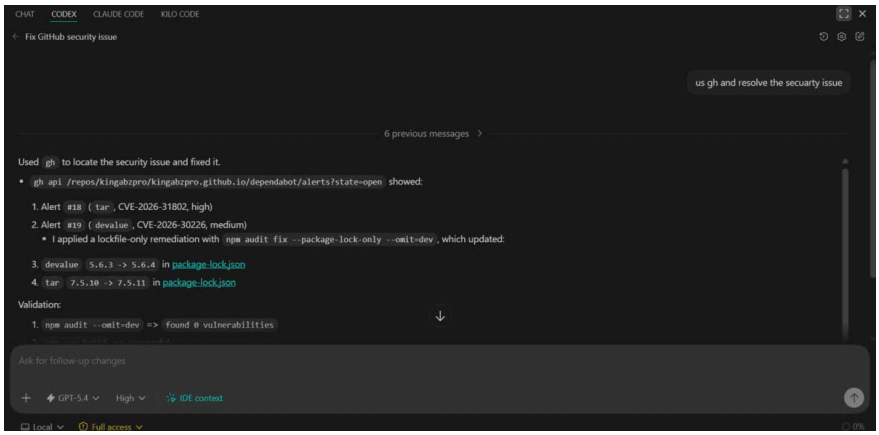
Shell tools are one of the elements that help the Codex feel like a 'true AI coding agent' rather than just a code generator.

The current workflows of the Codex CLI and IDE revolve around this idea. The Codex can read files, edit code, and run commands directly within the project. OpenAI also recommends using shell tools for terminal tasks.

This is especially important because much of the practical software engineering work takes place within the CLI, from the GitHub CLI (gh) to deployments with Vercel or other local tools.

This also helps reduce reliance on complex abstraction layers like MCP servers or unnecessary custom skills. Instead of creating multiple intermediate layers, you can instruct the Codex to directly use CLI tools already in your daily workflow. The result is usually:

1. fewer tokens
2. faster processing speed
3. The workflow is closer to a real-world local development environment.
4. by leveraging tools that programmers are already familiar with and trust.



Many people have used the Codex in Visual Studio Code almost daily for both personal and professional projects, and have seen remarkable results. Over time, the Codex has become increasingly powerful, sometimes giving them a feeling of being an 'imposter' as the AI ??fixes errors in just a few minutes.

However, the key is not to use the Codex haphazardly, but to learn how to work with it properly. The biggest differentiating factors include writing clearer documentation, carefully managing context, using Planning Mode for major changes, and creating custom skills for repetitive workflows.

Furthermore, requiring the Codex to verify results by running tests, checking the website or UI with tools like Playwright, and combining CLI and shell tools to directly interact with the local environment also significantly improves workflow smoothness.

All of these factors make the experience of working with the Codex increasingly feel like collaborating with a real AI coding agent rather than just using a simple code generation tool.