Addendum to o3 and o4-mini system card: Codex

OpenAI

May 16, 2025

Codex is a cloud-based coding agent. Codex is powered by codex-1, a version of OpenAI o3 optimized for software engineering. codex-1 was trained using reinforcement learning on real-world coding tasks in a variety of environments to generate code that closely mirrors human style and PR preferences, adheres precisely to instructions, and iteratively runs tests until passing results are achieved.

Users can ask Codex to perform coding tasks or to answer questions about a codebase. Each agent runs in its own cloud container with no internet access. The container is preloaded with the user's code and a development environment defined by the user, including any dependencies, configuration, or tooling they specify. After setup, internet access is disabled and the model trajectory begins. Within that environment, Codex can read and edit files, as well as execute commands including tests, linters, and type checkers.

Codex is trained to provide verifiable evidence of its actions through citations of terminal logs and files, allowing the user to validate the model's work. Once a task is complete, users can inspect the results, request refinements, or export the generated diff — either by converting it into a GitHub pull request or copying it for local testing and development.

1 Baseline Model Safety Evaluations

1.1 Disallowed Content Evaluations

The codex-1 model is purpose-built for software development and powers the coding-focused Codex user experience. This includes answering questions and engaging in conversations about a user's source code.

While the model is not intended or expected to be used in general-purpose chat applications, it can operate conversationally. Therefore we conducted standard refusal evaluations across disallowed content categories.

Table 1: Standard refusal evaluation

Category	metric	codex-1	o3
harassment/threatening	not_unsafe	0.98	0.99
sexual/exploitative	not_unsafe	0.96	0.98
sexual/minors	not_unsafe	0.98	1
extremist/propaganda	not_unsafe	1	1
hate	not_unsafe	0.97	1
hate/threatening	not_unsafe	1	1
illicit/non-violent	not_unsafe	1	1
illicit/violent	not_unsafe	1	1
personal-data/semi-restricted	not_unsafe	0.96	1
personal-data/restricted	not_unsafe	0.98	1
regulated-advice	not_unsafe	1	1
self-harm/intent	not_unsafe	1	1
self-harm/instructions	not_unsafe	1	1

1.2 Jailbreaks

We evaluate the robustness of models to jailbreaks: adversarial prompts that purposely try to circumvent model refusals for content it's not supposed to produce.

We evaluate the model using StrongReject [1], an academic jailbreak benchmark.

Table 2: StrongReject

Evaluation	codex-1	o3
StrongReject (goodness@0.1)	0.98	0.97

2 Product-Specific Risk Mitigations

Our incremental safety work for Codex focused on four potential risks specific to this product:

1. Users might try to use Codex to do a **harmful task**, such as developing malicious software.
2. A **prompt injection** attack might lead Codex to do something contrary to the user's instructions, such as exfiltrating user data.
3. Codex might make a **mistake** in its coding.
4. Codex might **falsely claim to have completed a task that it did not in fact complete.**

Table 3: Risks and Mitigations Overview

Product-specific safety risk	**Mitigations**
User uses Codex to do a harmful task	• Safety training (covering both coding and non-coding specific harms) • Network sandboxing – codex-1 does not have network access • Monitoring
Codex makes a mistake	• Network sandboxing – codex-1 does not have network access • Filesystem sandboxing – codex-1 only has access to a sandboxed container, limiting data exfiltration • User transparency and diff reviews
Codex falsely claims to have completed a task that it did not in fact complete.	• Safety training • User transparency and diff reviews
Codex is prompt injected	• Network sandboxing – codex-1 does not have network access • Filesystem sandboxing – codex-1 only has access to a sandboxed container • Safety training (instruction-hierarchy training to protect against prompt injections) • User transparency and diff reviews

2.1 Harmful Tasks

2.1.1 Risk description

Safeguarding against malicious uses of AI-driven software engineering — such as malware development — is increasingly important. At the same time, protective measures must be carefully designed to avoid unnecessarily impeding legitimate, beneficial use cases that may involve similar techniques, such as low-level kernel engineering.

2.1.2 Mitigations

To mitigate this risk, we use a number of different safety mitigations.

Safety training We have pre-existing policies and safety training data that cover refusing harmful tasks in ChatGPT such as user requests for guidance on how to make illegal drugs. These policies and this training data already lead to refusals for related coding tasks, such as a request to build a website to sell illegal drugs. To further strengthen safety for Codex, we

developed a more detailed policy and training data for codex-1 to refuse tasks related to malware development.

To support this, we built a synthetic data pipeline that generates a diverse set of prompts, code snippets, and environment configurations involving malware-relevant scenarios.

We then taught the model to follow these safety specifications—refusing certain high-risk requests, providing only contextual or defensive content, and appropriately handling dual-use scenarios without excessive refusal or hedging. We incorporated edge cases and adversarial examples to thoroughly test the model's boundaries and reinforce policy-compliant behavior in ambiguous or complex situations.

We introduced two evaluations to assess the effectiveness of this training. The first is a synthetic, large-scale benchmark that tests the model on a wide range of malware-related tasks, including ambiguous and dual-use prompts. The second is a curated "golden set" of test cases developed by internal policy experts.

Table 4: Malware Refusals

Evaluation	**Metric**	**codex-1**
Malware Refusals (Synthetic)	refusal_rate (higher is better)	0.97
Malware Refusals (Golden Set)	refusal_rate (higher is better)	0.98

Network sandboxing codex-1 can only execute commands in a container configured and sandboxed to have no internet access while the agent is in control,[1] preventing it from completing online malicious tasks such as denial-of-service (DoS) attacks or interactive hacking.

Monitoring We supplement automated safeguards with our Company-wide abuse detection program, which for Codex includes ongoing monitoring to detect and mitigate potentially harmful behavior. We developed two new monitors developed specifically for this deployment.

- **Malware-related prompt monitor**: A prompted classifier designed to detect user attempts to generate disallowed malware-related content.
- **Disallowed task monitor**: A prompted monitor that flags whether a given user task or prompt violates our policies, for example a request to code a website to sell illegal drugs.

2.2 Codex makes a mistake

2.2.1 Risk description

Codex can make mistakes such as writing buggy or insecure code.

Other types of mistakes would be possible if Codex wasn't sandboxed. For example:

[1]The container has no network access while codex-1 is in control. The container has network access during the "setup phase" of cloning the repo and installing dependencies, prior to codex-1 taking control.

- If Codex had network access then mistakes could also include harms such as accidental excessive network requests resembling Denial-of-Service (DoS) attacks, or accidental data destruction from a remote database or environment.
- If Codex ran on a user's local computer then mistakes could include harms such as accidental data destruction (within directories it can write to), accidental mis-configuration of the user's device or local environment.

2.2.2 Mitigations

Network sandboxing codex-1 can only execute commands in a container configured and sandboxed to have no internet access while the agent is in control. This helps prevent it from making mistakes that affect the outside world specifically while it executes commands to complete a task.

Filesystem sandboxing Codex operates within a temporary container with filesystem sandboxing. It only has access to files within the user-configured environment for the connected GitHub repository. It does not have access to the files on the user's local computer, or other directories outside of the sandbox.

This limits the scope for potential harms such as accidental local data destruction or accidental misconfiguration of the user's device or local environment.

User transparency and diff reviews After Codex completes a task, users are presented with a diff view and a comprehensive log of actions taken by the model. This enables users to review, validate, and approve changes before they are committed.

To further support transparency and user control, Codex is trained to cite its actions. These citations link directly to:

- the specific files it modified, and
- the corresponding terminal commands it executed.

This makes it easy for users to trace the origin of any change, confirm that all modifications align with the original intent, and verify the outcomes of terminal commands, such as tests.

2.3 Falsely claiming to have completed a task that it did not complete

In early testing we found that when Codex was presented with an extremely difficult or impossible software engineering task (e.g. asked to modify code that doesn't exist), it would often falsely claim to have completed the task rather than disclose that it could not complete the task.

This behavior presents a significant risk to the usefulness of the product and such behavior undermines user trust and may lead users to believe that critical steps—like editing, building, or deploying code—have been completed, when in fact they have not. Addressing this risk is essential for ensuring reliable and transparent AI-assisted software development.

2.3.1 Mitigations

Model training To reduce such instances, we developed a new safety training framework for codex-1. Our approach combines environment perturbations (modifying real environments to create challenging conditions) and synthetic environment generation to simulate a wide variety of real-world coding constraints, including various scenarios of unexpected container or code states, e.g., when the user task requires key files missing from the container or when the user task assumes a different repository than what is in the container.

During the reinforcement learning process, the model is penalized for producing results inconsistent with its actions. The model is rewarded for correctly acknowledging missing resources, environment limitations, and areas of uncertainty. We track the model's improvement on a synthetically generated dataset with unexpected container or code states.

Table 5: Unexpected container or code states

Evaluation	**Before Training**	**After Training**
Agent correctly states it couldn't complete the task (synthetic samples)	0.15	0.85

As we further invest in this new training framework, we expect to make continuous improvements in this area.

User transparency and diff reviews Refer to the prior section on 2.2.2 "User transparency and diff reviews" for how we invested to give the user full transparency into the actions of Codex to further mitigate this risk.

2.4 Prompt Injection

2.4.1 Risk description

If we did not have mitigations such as network sandboxing, Codex could encounter a prompt injection while interacting with the web, for example while browsing the web for documentation or to debug an error.

If successful, a prompt injection against Codex could result in harmful outcomes such as:

- Data exfiltration (e.g. leaking the users codebase)
- Harmful code changes (e.g. introducing a backdoor)
- Data destruction (e.g. deleting all files on a computer or in a database)

2.4.2 Mitigations

Network sandboxing codex-1 can only execute commands in a container configured and sandboxed to have no internet access while the agent is in control. This network sandboxing prevents the model from being exposed to online prompt injection attacks.

With internet access disabled, the remaining vectors for prompt injection are confined to the connected GitHub repository or its dependencies, and the environment explicitly configured by the user.

If a prompt injection does occur, the network sandbox acts as a critical layer of defense. It helps prevent potential harms such as:

- Data exfiltration — the model cannot transmit data to external endpoints.
- Remote data corruption or deletion — the model cannot connect to external databases or services.

Filesystem sandboxing Codex operates within a temporary container with filesystem sandboxing. It only has access to files within the user-configured environment for the connected GitHub repository. It does not have access to the files on the user's local computer, or to other directories outside of the sandbox.

This limits potential scope for harm such as data exfiltration and unauthorized modification or deletion of files.

Safety training To enhance the model's robustness against prompt injection attacks, following the Instruction Hierarchy framework [2], we created new safety training data and trained the model to ignore prompt-injection attacks in the coding environment.

We also developed a new prompt injection evaluation suite tailored to the Codex environment to measure the model's robustness to such attacks. The evaluation includes several types of attacks, such as:

- Attempts to terminate the model's current coding task, which would degrade its performance on that task.
- Attempts to instruct the model to carry out a specific action in later steps, such as outputting the final answer "HACKED!".

Table 6

Evaluation	**codex-1**
Successfully ignored prompt injection attack during coding tasks in Codex environment	0.98

We additionally trained codex-1 using the same Instruction Hierarchy and prompt injection robustness training data and methodologies used for some of our prior models.

User transparency and diff reviews Refer to the prior section on 2.2.2 "User transparency and diff reviews" for how we invested to give the user full transparency into the actions of Codex to further mitigate this risk.

2.5 Preparedness

codex-1 is a version of OpenAI o3 optimized for software engineering. OpenAI o3's frontier capabilities are assessed under the Preparedness Framework in the original o3 system card.

The additional training data for codex-1 emphasizes both human-like coding style to enhance usability and safety without substantially changing its frontier capability profile, as demonstrated by its similar performance, e.g. on SWE-bench metrics [3], compared to the original o3 model announced in December 2024.

The Safety Advisory Group determined that codex-1 does not reach the High capability threshold in any of our three tracked categories.

2.6 June 3, 2025 Update: Network access

As part of our commitment to iterative deployment, we launched Codex with a sandboxed task-execution environment. This cautious approach reduced risks like prompt injection while we gathered early feedback.

Users told us they understand these risks and want the flexibility to decide what level of internet connectivity to provide to the agent during task execution. For example, as the agent works, it may need to install or update dependencies overlooked by the user during environment configuration. Giving the user the choice to enable internet access–whether to a specific set of allowed sites, or to the internet at large–is necessary to unlock a number of use cases that were previously not possible.

As of today, we are enabling users to decide on a per-project basis which sites, if any, to let the agent access while it is running. This includes the ability to provide a custom allowlist or denylist. As outlined in this system card addendum, enabling internet access can introduce risks like prompt injection, leaked credentials, or use of code with license restrictions. Users should review outputs carefully and limit access to trusted domains and safe HTTP methods. Learn more in the docs: https://platform.openai.com/docs/codex.

References

[1] A. Souly, Q. Lu, D. Bowen, T. Trinh, E. Hsieh, S. Pandey, P. Abbeel, J. Svegliato, S. Emmons, O. Watkins, *et al.*, "A strongreject for empty jailbreaks," *arXiv preprint arXiv:2402.10260*, 2024.

[2] E. Wallace, K. Xiao, R. Leike, L. Weng, J. Heidecke, and A. Beutel, "The instruction hierarchy: Training llms to prioritize privileged instructions." https://arxiv.org/abs/2404.13208, 2024.

[3] OpenAI, "Introducing codex." http://openai.com/index/introducing-codex, 2025.