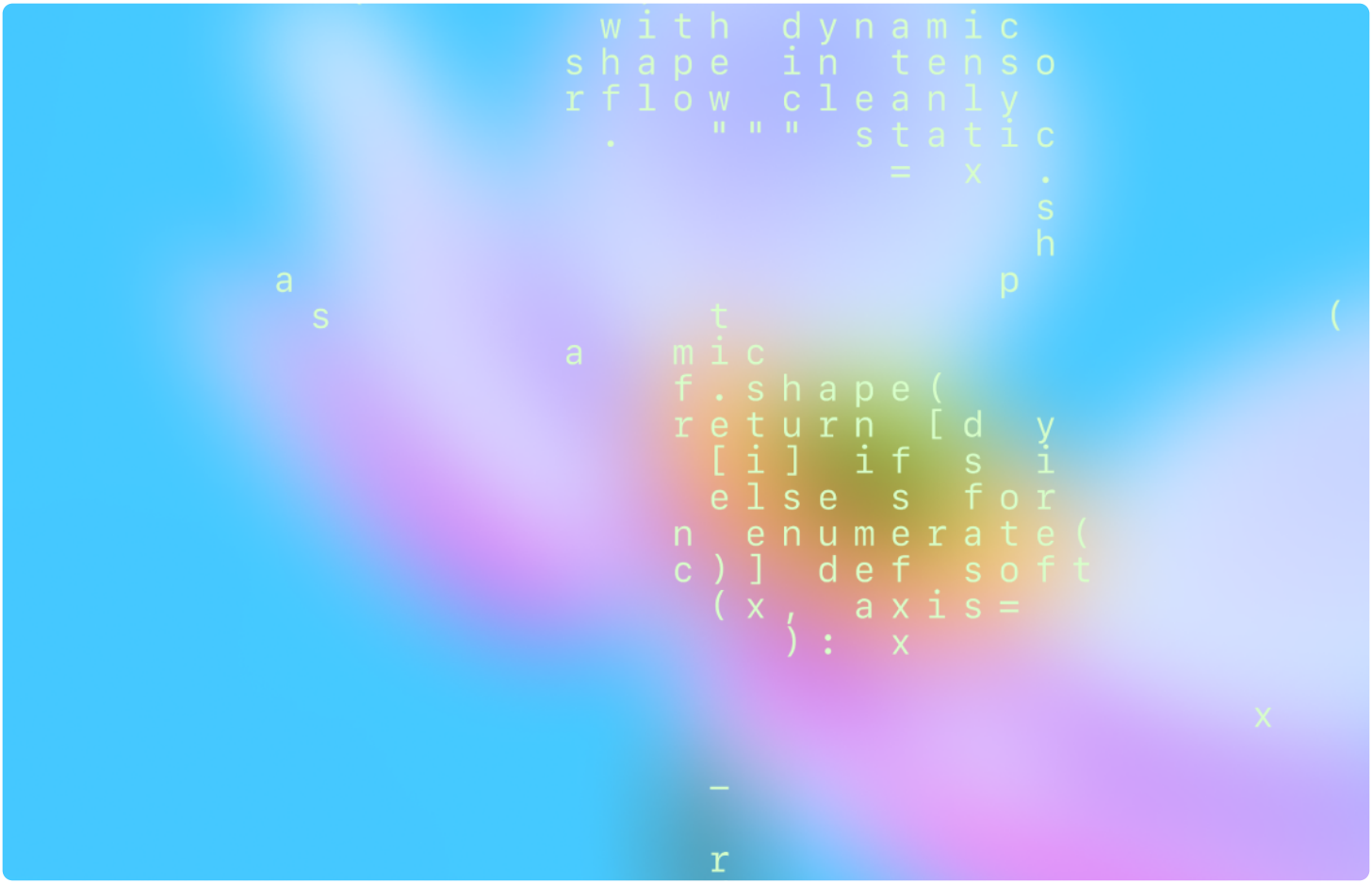
OpenAI

OpenAI における Codex 活用方法



目次

はじめに

Codex は、OpenAI の Security（セキュリティ）、Product Engineering（プロダクト開発）、Frontend（フロントエンド）、API、Infrastructure（インフラ）、Performance Engineering（パフォーマンス最適化） など、社内のさまざまなチームで日常的に使用されています。各チームは、複雑なシステムの理解や大規模コードベースのリファクタリングから、新機能のリリース、厳しい期限の中でのインシデント対応まで、幅広いエンジニアリングタスクを加速するために Codex を活用しています。

OpenAI のエンジニアへのインタビューと社内の利用データに基づき、Codex がチームの開発スピードを高め、仕事の品質を向上させ、大規模な複雑性を扱う上でどのように役立つかを示すユースケースとベストプラクティスをまとめました。

コードの理解

Codex は、プロジェクトへの新規参画時や、デバッグ時、インシデント調査時に、これまで十分に把握できていなかったコードベースの領域についても、チームが短時間で状況を把握できるよう支援します。

各チームは Codex を使って、ある機能のコアロジックを特定したり、サービスやモジュール間の関係をマッピングしたり、システム全体のデータフローを追跡したりすることがよくあります。 また、Codex は、通常であれば大きな手作業が必要となる、アーキテクチャパターンや不足しているドキュメントの洗い出しにも役立ちます。

インシデント対応中、Codex はコンポーネント間の相互作用を可視化したり、障害の影響がシステム全体にどのように波及するかを追跡したりすることで、エンジニアが新しい領域にも素早く対応できるよう支援します。

OpenAI 社内チームの声

バグを修正するときは、Ask モードを使って、同じ問題がコードベースのほかのどこで発生し得るかを確認します

パフォーマンスエンジニア、リトリーバルシステム

オンコールのときは、スタックトレースを貼り付けて、認証フローがどこにあるかを Codex に聞きます。そうすると、すぐに正しいファイルを特定してくれるので、素早くトリアージできます

サイト信頼性エンジニア、API プラットフォーム

Codex は、Terraform や Python にまたがるリポジトリに対する質問に、grep よりもはるかに速く答えてくれます

DevOps エンジニア、インフラストラクチャ

コード理解のために Codex を試す場合のサンプルプロンプト:

- [ ] このリポジトリで、認証ロジックが実装されている箇所を教えてください。
- [ ] このサービス内で、エントリポイントからレスポンスまでリクエストがどのように流れるかを要約してください。
- [ ] [モジュールの名前] と相互作用するモジュールと、障害時の処理の流れを教えてください。

リファクタリングおよびマイグレーション

Codex は、複数のファイルやパッケージにまたがる変更を行う場面でよく使用されます。たとえば API を更新するとき、パターンの実装方法を変更するとき、新しい依存関係へ移行するときなどに、Codex を使うことで変更を一貫して適用しやすくなります。

同じ更新を数十個のファイルに対して行う必要がある場合や、正規表現や検索・置換だけでは簡単に検出できない構造や依存関係を考慮して更新する必要がある場合に、特に有用です。

また、Codex はコードクリーンアップにも利用されています。たとえば、肥大化したモジュールを分割したり、古いパターンをモダンなパターンに置き換えたり、テストしやすくするためにコードを整えたりする場面です。

OpenAI 社内チームの声

> Codex は、既存のすべての `getUserById()` を新しいサービスパターンに置き換え、PR を作成しました。数時間かかる作業を数分でやり遂げたのです
>
> **バックエンドエンジニア、ChatGPT Web**

> ローンチを妨げている要因を解消するために、Codex に古いパターンのすべての出現箇所をスキャンさせ、その影響を Markdown で要約させてから、修正を含む PR を作成させています
>
> **プロダクトエンジニア、ChatGPT Enterprise**

リファクタリングやマイグレーションに Codex を試す場合のサンプルプロンプト:

- [ ] このファイルを関心ごとに別々のモジュールへ分割し、それぞれのモジュール用のテストを生成してください。
- [ ] すべてのコールバックベースのデータベースアクセスを async/await に変換してください。

パフォーマンス最適化

Codex はパフォーマンスボトルネックを特定し、対処するために使用されます。

チューニングや信頼性向上の作業中に、エンジニアは Codex に、低速またはメモリ集約的なコードパスを分析させます。たとえば非効率なループ、冗長な処理、高コストなクエリなどです。その上で、 Codex に最適化された代替案を提案させます。これにより、多くの場合、効率や信頼性が大きく改善します。

また Codex は、現在も実際に使われているリスクの高いパターンや非推奨パターンを特定し、コードの健全性を維持するためにも利用されています。 OpenAI のチームは、長期的な技術的負債を削減し、リグレッションを事前に防ぐために Codex を活用しています。

OpenAI 社内チームの声

> Codex を使って、高コストな DB 呼び出しの重複をスキャンしています。ホットパスを検出したり、後で自分でチューニングできるバッチクエリのたたき台を作成したりするのに、Codex はとても役立っています
>
> **インフラストラクチャエンジニア、API 信頼性**

> Codex はパフォーマンス問題を素早く見つけるのに非常に役立ちます。プロンプトに5分かけるだけで、作業時間を30分節約できます
>
> **プラットフォームエンジニア、モデル提供**

パフォーマンス最適化のために Codex を試す場合のサンプルプロンプト:

- [ ] このループをメモリ効率の観点から最適化し、提案のバージョンの方が高速になる理由を説明してください。
- [ ] このリクエストハンドラー内で繰り返し実行されている高コストな処理を見つけ、キャッシュの適用候補を提案してください。
- [ ] この関数内で DB クエリをバッチ処理する、より高速な方法を提案してください。

テストカバレッジの改善

Codex は、エンジニアがテストを書く速度を向上させるのに役立ちます。特にカバレッジが薄い箇所や、テストがまったく存在しない箇所で効果を発揮します。

バグ修正やリファクタリングに取り組む際、エンジニアは Codex に、エッジケースや失敗につながりやすいパスをカバーするテストの提案を求めることがよくあります。新しいコードに対しては、関数シグネチャや周辺のロジックに基づいて、ユニットテストや統合テストを生成できます。

Codex は、空の入力や最大長、または初期テストでは見落とされがちな、例外的だが仕様上は許容されている状態といった境界条件を特定するのに特に有用です。

OpenAI 社内チームの声

カバレッジの低いモジュールに Codex を一晩走らせておくだけで、朝にはそのまま実行可能なユニットテストの PR が作成されています

フロントエンドエンジニア、ChatGPT デスクトップ

モノレポのブランチ切り替えがつらいときは、今いるブランチで作業を続けながら、*Codex* にテストを書かせて *CI* を走らせています

バックエンドエンジニア、支払いと請求

テストカバレッジ向上のために Codex を試す場面のサンプルプロンプト:

- [ ] この関数に対して、エッジケースおよび失敗パスを含む単体テストを書いてください。
- [ ] このソートユーティリティに対する property-based テストを生成してください。
- [ ] このテストファイルを拡張し、null 入力や不正な状態に関する不足しているシナリオをカバーしてください。

開発速度の向上

Codex は、開発サイクルの開始と終了の両方を加速することで、チームの開発スピード向上を支援します。

新機能を立ち上げるとき、エンジニアは Codex を使ってボイラープレートを生成します。フォルダやモジュール、API スタブを自動で用意し、すべてを手作業で配線せずに、すぐに実行可能なコードを立ち上げます。

プロジェクトがリリース段階に近づくと、Codex はバグのトリアージ、最終段階の実装の抜け漏れ対応、ロールアウトスクリプトやテレメトリフック、設定ファイルの生成といった、小さいが重要なタスクを担うことで、厳しい納期の達成を支援します。

Codex は、プロダクトのフィードバックをスターターコードに変換する用途にも使われています。エンジニアは、ユーザーからのリクエストや仕様書をそのまま入力し、Codex にラフなドラフトコードを生成させてから、後で改良していくことがよくあります。

OpenAI 社内チームの声

一日中ミーティングに参加していたのに、Codex がバックグラウンドで動いていたおかげで、PR を4件マージできました

プロダクトエンジニア、ChatGPT Enterprise

Codex のおかげで、バックログに埋もれていたであろう優先度の低い3〜4件の修正を、完璧な形でリリースできました。とても力づけられる体験でした

フルスタックエンジニア、社内ツール

Codex による開発速度向上を試す場合のサンプルプロンプト:

- [ ] 基本的なバリデーションとロギングを備えた POST /events 用の新しい API ルートを生成してください。
- [ ] このテンプレート使って、新しいオンボーディングフローの成功 / 失敗をトラッキングするためのテレメトリフックを生成してください。
- [ ] 以下の仕様に基づいてスタブ実装を作成してください。[スペックを挿入]。

作業の流れを維持する

Codex は、エンジニアのスケジュールが細切れになり、割り込みが多い状況でも、生産性を維持できるよう支援します。

これは、未完了の作業を記録したり、メモを動作するプロトタイプに変換したり、後から再訪できる探索的なタスクを派生させたりするために使われます。これにより、特にオンコール対応中や会議が多いときでも、作業の前提や経緯を保ったまま作業を一時停止し、再開しやすくなります。

OpenAI 社内チームの声

もしついでに直せるような小さな修正を見つけたら、ブランチを切り替える代わりに Codex にタスクとして任せておき、手が空いたときにその PR をレビューします

バックエンドエンジニア、 ChatGPT API

Slack のスレッドや Datadog のトレース、issue などを日常的に Codex に共有して、高優先度の作業に集中できるようにしています

API エンジニア、インフラストラクチャ監視

作業の流れを維持するために Codex を試す場合のサンプルプロンプト:

- [ ] このサービスをリファクタリングしてより小さなモジュールに分割するための計画を作成してください。
- [ ] リトライロジックをスタブ化して TODO を追加してください。バックオフロジックは後で私が記述します。
- [ ] このファイルを要約して、明日どこから作業を再開すればよいか分かるようにしてください。

調査とアイデア創出

Codex は、代替ソリューションを探したり、設計上の判断を検証したりするような、正解が1つに定まらない作業にも有用です。問題を解くさまざまな方法を提示させたり、なじみのないパターンを探索したり、前提条件を妥当かどうか検証したりできます。これにより、トレードオフが明確になり、設計の選択肢が広がり、実装の判断をより精緻にできます。

また、Codex は関連するバグを特定するためにも使われます。既知の問題や非推奨のメソッドがある場合、Codex はコード内の他の箇所の類似パターンを特定できます。これにより、リグレッションの発見や、クリーンアップ作業の完了がしやすくなります。

OpenAI 社内チームの声

Codex はコールドスタート問題の解決に役立ちます。仕様書やドキュメントを貼り付けると、コードを生成してくれたり、自分が見落としていた点を示してくれたりします

プロダクトエンジニア、ChatGPT デスクトップ

バグを修正した後、どこに類似のバグが潜んでいる可能性があるかを *Codex* に尋ねてから、フォローアップのタスクを作成します

パフォーマンスエンジニア、検索・取得システム

調査やアイデア創出に Codex を試す場合のサンプルプロンプト

- [ ] このシステムがリクエスト/レスポンスではなくイベント駆動型だった場合、どのように動作するか説明してください。
- [ ] クエリビルダーを使わずに、手動で SQL 文字列を組み立てているモジュールをすべて見つけてください。
- [ ] より関数型スタイルになるように書き直してください。ミューテーションと副作用は避けてください。

ベストプラクティス

Codex は、構造や文脈が与えられ、反復して試行錯誤する余地があるときに、最も良く機能します。以下に、OpenAI のチームが日々の業務で Codex から安定して価値を引き出すために、実践しているポイントをご紹介します。

Ask モードから開始する

大きな変更を行う場合は、まず Ask モードで Codex に実装プランを作成するよう依頼し、そのプランを Code モードに切り替えた後のプロンプトとして活用します。この2ステップの手順により、Codex の前提が安定し、出力のエラーを避けやすくなります。 Codex は、スコープが明確なタスクで最もよく機能します。たとえば、ご自身やチームメイトが約1時間で完了できる、または数百行程度のコードで実装できる規模のタスクです。 モデルが改善されるにつれて、Codex が扱えるタスクの規模が大きくなることが期待できます。

Codex の開発環境を反復的に改善する

スタートアップスクリプト、環境変数、およびインターネットアクセスを設定すると、Codex のエラー率を大きく減らせます。タスクを実行するときは、Codex の環境設定で修正できるビルドエラーがないか確認してください。反復的な調整が必要になるかもしれませんが、長期的には大きな効率向上につながります。

GitHub issue を作成するつもりでプロンプトを書く

Codex は、PR や issue で変更内容を説明するときと同じような形でプロンプトを書いた方が、より的確に応答します。つまり、必要に応じてファイルパス、コンポーネント名、diff、ドキュメントのスニペットなどを含めるということです。「[module X] で行われているのと同じ方法でこれを実装して」といった書き方でプロンプトを入力すると、結果が向上します。

Codex タスクキューを軽量なバックログとして使用する

周辺的なアイデア、途中の作業、ついでに見つけた小さな修正を取りこぼさないように、タスクとして Codex にどんどん投げてみましょう。一度で完全な PR を作成する必要はありません。Codex は作業の一時的な仮置き場としても有用であり、集中できる状態に戻ったときに立ち返る場所として機能します。

AGENTS.md を使用して継続的な文脈を与える

リポジトリ内でプロンプト全体を通じて Codex がより効果的に動作できるようにするため、AGENTS.md ファイルを整備しましょう。これらのファイルには通常、命名規則、ビジネスロジック、既知の注意点や、コードだけからは Codex が推測できない依存関係などを含めます。AGENTS.md ファイルの構成方法についてはドキュメントで詳細をご確認ください。

「Best-of-N」を活用して出力を改善する

Best-of-N 機能を使うと、1つのタスクに対して複数の回答を同時に生成できます。これにより複数の解決策を素早く比較し、その中から最良のものを選択できます。より複雑なタスクでは、複数の試行結果を確認し、異なる回答の一部を組み合わせることで、より質の高い結果を得ることができます。

OpenAI