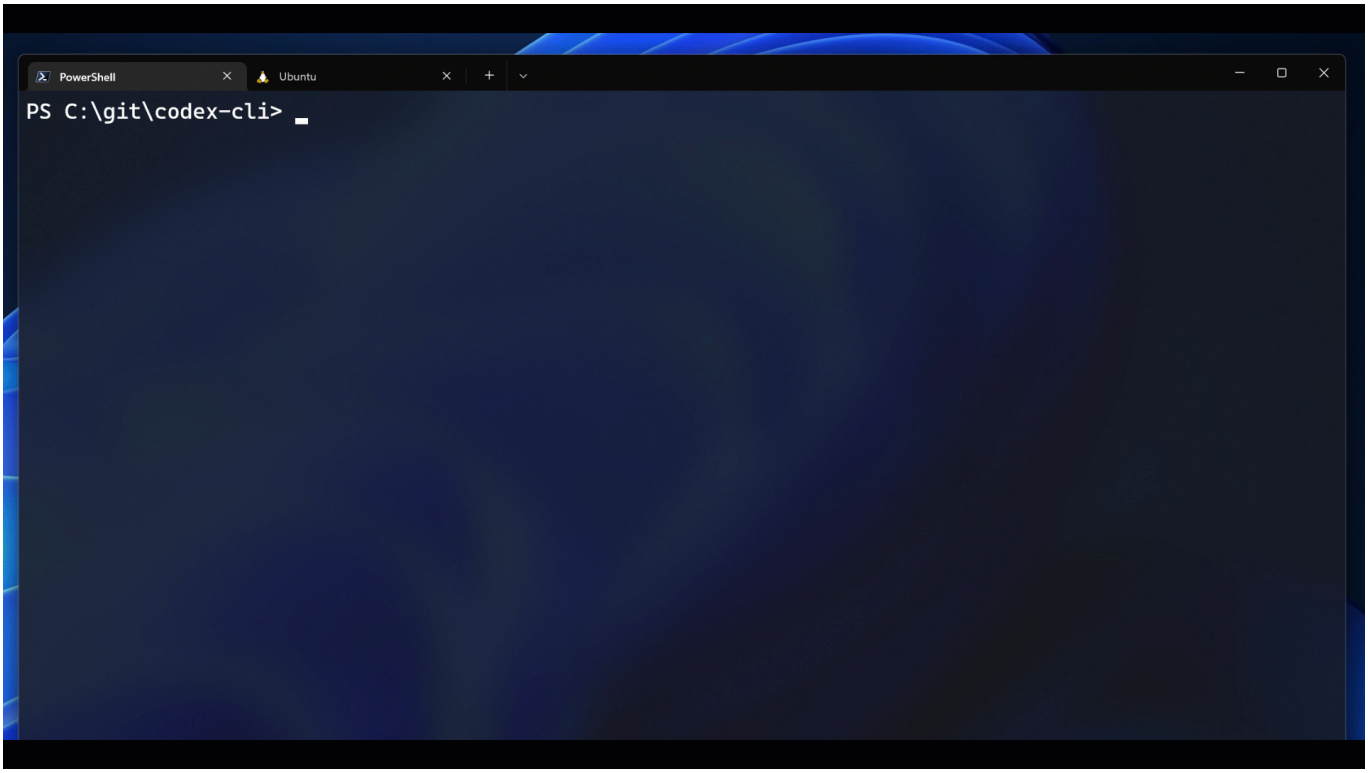
Codex CLI - 自然言語コマンドラインインターフェース

このプロジェクトはGPT-4oを使用して、自然言語コマンドをPowerShell、Z shellおよびBashのコマンドに変換します。



コマンドラインインターフェース（CLI）は、私たちが機械と対話するための最初の主要なユーザーインターフェースでした。CLIは非常に強力であり、ほぼすべてのことが可能ですが、ユーザーが自分の意図を非常に正確に表現する必要があります。ユーザーは「コンピューターの言語を知る」必要があります。

大規模言語モデル（LLM）の登場、特にコードに関して訓練されたモデルにより、自然言語（NL）を使用してCLIと対話することが可能になりました。実際、これらのモデルは自然言語とコードの両方を十分に理解しており、一方から他方へ変換することができます。

このプロジェクトは、クロスシェルのNL->Codeエクスペリエンスを提供し、ユーザーが自然言語を使って好みのCLIと対話できるようにすることを目的としています。ユーザーはコマンドを入力し（例：「私のIPアドレスは何？」）、`Ctrl + G`を押すと、使用しているシェルに適したコマンドの提案を得られます。このプロジェクトではGPT-4oモデルを使用しており、優れたコード生成能力を持っています。プロンプトエンジニアリングと呼ばれる手法（下記のセクション参照）を使用して、モデルから適切なコマンドを引き出しています。

注意：モデルは間違える可能性があります！理解できないコマンドは実行しないでください。コマンドの動作がわからない場合は、`Ctrl + C`を押してキャンセルしてください。

このプロジェクトはzsh_codexプロジェクトからの技術的なインスピレーションを得て、複数のシェルに対応できるよう機能を拡張し、モデルに渡すプロンプトをカスタマイズしています（下記のプロンプトエンジニアリングのセクションを参照）。

目的声明

このリポジトリは、Microsoft Build conference 2022をサポートするための実装例とリファレンスを提供することで、アプリケーションでのCodexの使用理解を深めることを目的としています。これはリリース製品として意図されたものではありません。したがって、このリポジトリではOpenAI APIに関する議論や新機能のリクエストは対象外です。

要件

- Python 3.7.1+
  - [Windows]: PythonがPATHに追加されていること。
- OpenAIアカウント
  - OpenAI APIキー
  - OpenAI Organization Id (省略可能。複数の組織がある場合のみ必要です)
  - OpenAIモデル名：最良の結果を得るには`gpt-4o`を使用してください。利用可能なモデルの確認についてはこちらを参照してください。

環境変数の設定

APIキーや組織IDなどの設定は以下の環境変数を使用して設定することもできます：

- `OPENAI_API_KEY` - OpenAI APIキー
- `OPENAI_ORG_ID` - OpenAI 組織ID (省略可能)
- `OPENAI_MODEL` - 使用するモデル名 (省略可能、デフォルトは`gpt-4o`)

環境変数が設定されていない場合は、セットアップスクリプトで入力したAPIキーがファイルから読み込まれます。

インストール

PowerShell、bash、zshのインストール手順はこちらを参照してください。

設定ファイル

Codex CLIは以下の設定ファイルを使用します：

~/.openai/codex-cli.json - API認証情報と設定を含む設定ファイル：

```
{
  "api_key": "YOUR_API_KEY",
  "organization": "YOUR_ORGANIZATION_ID",
  "model": "gpt-4o",
  "language": "en"  // 言語設定: "en"（英語）または "ja"（日本語）
}
```

`language`設定は、コマンド生成時に使用されるシステムプロンプトの言語を決定します。指定されていない場合、デフォルトで英語が使用されます。

使用方法

好みのシェル用に設定が完了したら、シェルにコメント（#で始まる）を書き込み、Ctrl + Gを押すことでCodex CLIを使用できます。

Codex CLIは主に2つのモード、シングルターンとマルチターンをサポートしています。

デフォルトでは、マルチターンモードはオフになっています。# start multi-turnと# stop multi-turnのコマンドを使用してオン/オフを切り替えることができます。

マルチターンモードがオンの場合、Codex CLIはモデルとの過去のやり取りを「記憶」し、以前のアクションやエンティティを参照できるようになります。例えば、Codex CLIでタイムゾーンをマウンテンに変更し、その後「パシフィックに戻して」と言うと、モデルは前回のやり取りから「それ」がユーザーのタイムゾーンであることを認識します：

```
# change my timezone to mountain
tzutil /s "Mountain Standard Time"

# change it back to pacific
tzutil /s "Pacific Standard Time"
```

このツールはcurrent_context.txtファイルを作成し、過去のやり取りを追跡して、各後続コマンドでモデルに渡します。

マルチターンモードがオフの場合、このツールはやり取りの履歴を追跡しません。マルチターンモードには長所短所があります - 文脈解決を可能にする一方で、オーバーヘッドも増加します。例えば、モデルが間違ったスクリプトを生成した場合、ユーザーはそれをコンテキストから削除したいと考えるでしょう。そうしないと、将来の会話ターンでも間違ったスクリプトが生成される可能性が高くなります。マルチターンモードをオフにすると、モデルは完全に決定論的に動作します - 同じコマンドは常に同じ出力を生成します。

モデルが一貫して間違ったコマンドを出力しているように見える場合は、# stop multi-turnコマンドを使用してモデルが過去のやり取りを記憶するのを停止し、デフォルトのコンテキストをロードできます。または、# default contextコマンドは、マルチターンモードをオンに保ちながら同様の効果を持ちます。

コマンド

コマンド	説明
start multi-turn	マルチターン体験を開始します
stop multi-turn	マルチターン体験を停止し、デフォルトコンテキストをロードします
load context <filename>	contextsフォルダからコンテキストファイルをロードします
default context	デフォルトのシェルコンテキストをロードします
view context	テキストエディタでコンテキストファイルを開きます
save context <filename>	コンテキストファイルをcontextsフォルダに保存します。名前が指定されていない場合は、現在の日時を使用します
show config	モデルとのインタラクションの現在の設定を表示します

コマンド	説明
`set <config-key> <config-value>`	モデルとのインタラクションの設定を変更します

setコマンドを使用してトークン制限、モデル名、温度を変更することで、体験を向上させることができます。例：`# set engine gpt-4o`、`# set temperature 0.5`、`# set max_tokens 50`。

プロンプトエンジニアリングとコンテキストファイル

このプロジェクトでは、自然言語からコマンドを生成するようGPT-4oを調整するために、「プロンプトエンジニアリング」と呼ばれる手法を使用しています。具体的には、NL->Commandsの一連の例をモデルに渡し、どのようなコードを書くべきかの感覚を与え、また使用しているシェルに適したコマンドを生成するよう促します。これらの例は`contexts`ディレクトリにあります。以下はPowerShellコンテキストの抜粋です：

```
# what's the weather in New York?
(Invoke-WebRequest -uri "wttr.in/NewYork").Content

# make a git ignore with node modules and src in it
"node_modules
src" | Out-File .gitignore

# open it in notepad
notepad .gitignore
```

このプロジェクトでは自然言語コマンドをコメントとしてモデル化し、モデルに期待するPowerShellスクリプトの例を提供しています。これらの例には、一行の補完、複数行の補完、マルチターンの補完（「open it in notepad」の例は前のターンで生成された`.gitignore`ファイルを参照）が含まれています。

ユーザーが新しいコマンド（例えば「what's my IP address」）を入力すると、そのコマンドをコンテキストに（コメントとして）追加し、それに続くコードを生成するようGPT-4oに依頼します。上記の例を見て、GPT-4oはコメントを満たす短いPowerShellスクリプトを書くべきだと理解するでしょう。

独自のコンテキストの構築

このプロジェクトには各シェル用のコンテキストと、その他の機能を備えたボーナスコンテキストがプリロードされています。これらに加えて、モデルから他の動作を引き出すための独自のコンテキストを構築できます。例えば、Codex CLIにKubernetesスクリプトを生成させたい場合、コマンドの例とモデルが生成する可能性のある`kubectl`スクリプトを含む新しいコンテキストを作成できます：

```
# make a K8s cluster IP called my-cs running on 5678:8080
kubectl create service clusterip my-cs --tcp=5678:8080
```

コンテキストを`contexts`フォルダに追加し、`load context <filename>`を実行してロードします。`src\prompt_file.py`内のデフォルトコンテキストを自分のコンテキストファイルに変更することもできます。

GPT-4oは例がなくても正しいスクリプトを生成することがよくあります。大量のコードで訓練されているため、特定のコマンドの生成方法を知っていることが多いです。ただし、独自のコンテキストを構築することで、求めている特定の種類のスクリプト（長いか短いか、変数を宣言するかどうか、以前のコマンドを参照するかどうかなど）を引き出すのに役立ちます。また、自分のCLIコマンドやスクリプトの例を提供して、GPT-4oが使用を考慮すべき他のツールを示すこともできます。

重要なことは、新しいコンテキストを追加する場合、マルチターンモードをオンにして自動デフォルト設定（エクスペリエンスが壊れることを防ぐために追加された機能）を避けることです。

例として、テキスト読み上げタイプのレスポンスを提供するCognitive Services APIを使用したcognitive services contextを追加しました。

トラブルシューティング

`DEBUG_MODE`を使用して、標準入力の代わりにターミナル入力を使用し、コードをデバッグします。これは新しいコマンドを追加する際やツールが応答しない理由を理解する際に役立ちます。

`openai`パッケージがツールでキャッチされないエラーをスローすることがあります。この場合、`codex_query.py`の最後にその例外用のcatchブロックを追加し、カスタムエラーメッセージを表示できます。

よくある質問

利用可能なOpenAIモデルを確認する方法

OpenAI組織ごとに異なるOpenAIモデルにアクセスできる可能性があります。利用可能なモデルを確認するには、List models APIを使用できます。以下のコマンドを参照してください：

- Shell

```
curl https://api.openai.com/v1/models \
  -H 'Authorization: Bearer YOUR_API_KEY' \
  -H 'OpenAI-Organization: YOUR_ORG_ID'
```

- PowerShell

  PowerShell v5（Windowsに付属のデフォルトバージョン）

```
(Invoke-WebRequest -Uri https://api.openai.com/v1/models -Headers
@{"Authorization" = "Bearer YOUR_API_KEY"; "OpenAI-Organization" =
"YOUR_ORG_ID"}).Content
```

  PowerShell v7

```
(Invoke-WebRequest -Uri https://api.openai.com/v1/models -Authentication
Bearer -Token (ConvertTo-SecureString "YOUR_API_KEY" -AsPlainText -Force) -
Headers @{"OpenAI-Organization" = "YOUR_ORG_ID"}).Content
```

Azureでサンプルを実行できますか？

サンプルコードはOpenAI APIのGPT-4oで使用できます。また、Azure経由でGPT-4oにアクセスできる場合は、Azure OpenAI Serviceでも使用できます。