
Explorando a API da OpenAI

Asimov Academy

ASIMOV

Conteúdo

01. Bem-vindos ao curso Explorando a API da OpenAI	4
02. Explorando a documentação da API	5
O que é uma API?	5
A OpenAI	6
Acessando a documentação da API	6
Capabilities	7
Modelos	7
Exemplos de prompts	8
Cookbook	8
Guias	9
03. Adicionando saldo a sua conta da OpenAI	10
Limites de uso gratuito	10
Adicionando saldo	11
Adicionando limites de gasto mensal	13
Verificação de uso	14
Custos dos modelos	15
04. Inicializando a biblioteca da OpenAI	17
Instalando a biblioteca	17
Gerando uma API Key	17
Inicializando o cliente	19
05. Utilizando o ChatGPT para gerar textos	21
Escrevendo seu primeiro script de geração de texto	21
Analizando o script inicial	22
Parâmetro messages	22
Analisando a classe de resposta do modelo	23
Analisando a mensagem de resposta	24
Demais parâmetros do método chatcompletion	25
06. Gerando uma stream de texto	28
07. DESAFIO - Criando um Chatbot em Python	30
08. Adicionando funções e ferramentas externas	32
Definição de funções externas	34

Adicionando especificações das ferramentas	35
Chamando <i>chat completion</i> com ferramentas externas	36
Entendendo o parâmetro <code>tool_calls</code> da resposta do modelo	36
Rodando as funções externas	37
Analisando a lista de mensagens final	39
09. DESAFIO - ChatBot Finanças	40
11. Fine-Tuning - otimizando um modelo	43
O que é Fine-Tuning?	43
Quando utilizar Fine-Tuning?	43
Usos comuns	44
Criando um modelo com Fine-Tuning em Python	44
Definição do problema	44
Preparação de dados	44
Criando modelo com Fine-Tuning	46
Verificando status do novo modelo	46
Utilizando o novo modelo	47
Comparando o modelo com Fine-Tuning e o modelo padrão	48
13. Apresentando a Assistants API para geração de texto avançada	50
Criando um Assistant	50
Criando um Thread	51
Adicionando mensagem a Thread	51
Solicitando ao Assistant para rodar uma Thread	52
Esperando a Thread rodar	52
Verificando a resposta	52
Analisando os passos de processamento do Assistant	53
14. Analisando dados com Assistants Code Interpreter	56
Enviando arquivos para o Assistant	56
Verificando os passos	58
Gerando gráficos com Assistants	60
15. Analisando arquivos pdf com Assistants Retrieval	63
16. Criando e editando imagens com Dall-e	66
Criando uma imagem	66
Salvando a imagem gerada	67
Visualizando a imagem	67

Editando uma imagem	68
Salvando a imagem gerada	71
Visualizando a imagem	71
Criando variações	72
Salvando a imagem gerada	73
Visualizando a imagem	73
17. Visão computacional com GPT-Vision	75
Interpretando uma imagem da internet	75
Interpretando uma imagem do seu computador	76
Interpretando palavras escritas	78
18. Criação de áudios a partir de textos	80
19. Transcrição de áudios	82
20. Mini-projeto - Chatbot com reconhecimento de fala	85
21. Finalizando o curso	87

01. Bem-vindos ao curso Explorando a API da OpenAI

Olá e bem-vindos ao nosso curso “Explorando a API da OpenAI”! Estou super animado por ter vocês aqui e mal posso esperar para explorarmos juntos esse universo fascinante da programação e da inteligência artificial.

Vocês já devem ter percebido que a IA está mudando o jogo em muitos campos, certo? É incrível como ela está transformando a maneira como lidamos com informações e realiza tarefas que antes pareciam exclusivas da mente humana. E o mais legal é que não estamos apenas assistindo a essa mudança; vamos fazer parte dela!

Com a API da OpenAI, temos a chance de interagir com alguns dos modelos de IA mais avançados que existem. E ao conciliarmos o poder desses modelos com as capacidades da programação Python, damos um salto para o futuro.

Ao longo deste curso, vamos aprender a usar essa API para realizar tarefas incríveis, como criar textos, gerar imagens e transcrever áudios. E o mais bacana é que vamos fazer tudo isso de um jeito bem prático, escrevendo nosso próprio código e vendo a mágica acontecer na frente dos nossos olhos.

Python é a ferramenta perfeita para isso. Ela é simples, direta e nos permite focar no que realmente importa: criar aplicações incríveis. E quando você combina essa simplicidade com o poder da IA, as possibilidades são praticamente infinitas.

Então, se vocês estão tão empolgados quanto eu para começar a explorar o que a IA pode fazer e como podemos usá-la para expandir nossas habilidades de programação, vocês estão no lugar certo. Vamos aprender, nos impressionar e, quem sabe, até criar algo que nunca imaginamos ser possível.

Sejam bem-vindos! Vamos nessa juntos e ver até onde podemos chegar com a API da OpenAI e nosso querido Python.

02. Explorando a documentação da API

Antes de iniciarmos nosso curso, é crucial compreender a estrutura do mesmo e saber onde encontrar informações em caso de dúvidas ou problemas. Para isso, vamos explorar o site da OpenAI e entender a organização da documentação. Dessa forma, você estará equipado para buscar informações de maneira autônoma, preenchendo quaisquer lacunas que possam surgir.

No entanto, antes de mergulharmos nesse processo, considero importante definirmos o que é uma API.

O que é uma API?

Uma API, sigla para Interface de Programação de Aplicações (em inglês, “Application Programming Interface”), consiste em um conjunto de regras e definições projetadas para permitir a comunicação entre softwares distintos. Ela atua como um intermediário, estabelecendo como os desenvolvedores podem solicitar informações ou dados de um programa, sem a necessidade de compreender os detalhes internos de como esses serviços são implementados.

As APIs desempenham um papel crucial no desenvolvimento contemporâneo de software, pois facilitam a integração e possibilitam a criação de soluções que combinam funcionalidades de várias fontes de maneira eficiente e escalável.

Imagine que você está em um restaurante com um cardápio repleto de opções de pratos. A cozinha representa um sistema complexo que você, como cliente, não vê ou com o qual interage diretamente. Nesse cenário, a API é semelhante ao garçom, que atua como o intermediário entre você e a cozinha. Você informa ao garçom (API) o que deseja comer (a solicitação de serviço ou dados), e ele comunica seu pedido à cozinha, onde o prato é preparado (o processamento interno do sistema). Em seguida, o garçom entrega o prato pronto a você (a resposta da API). Não é necessário entender como o prato é feito, quais ingredientes são utilizados ou o funcionamento interno da cozinha. O essencial é saber o que consta no cardápio (a documentação da API) e como fazer o pedido (como utilizar a API).

No contexto dos modelos de linguagem de grande escala (LLM), uma API é frequentemente disponibilizada por provedores como a OpenAI para simplificar o acesso aos modelos. Por exemplo, uma API pode permitir que você envie um texto e, em resposta, receba uma continuação coerente desse texto gerada pelo modelo, uma resposta a uma pergunta ou uma tradução. Isso possibilita que desenvolvedores, mesmo aqueles com conhecimentos básicos de programação como você, integrem funcionalidades de inteligência artificial avançadas em seus próprios aplicativos sem a necessidade de construir e treinar seus próprios modelos. Tal processo exigiria não apenas recursos computacionais significativos, mas também expertise especializada.

E a utilização dessas APIs que serão o foco do nosso curso!

A OpenAI

Nosso objetivo é explorar a API da OpenAI, a maior desenvolvedora de ferramentas de inteligência artificial do mundo na atualidade. O modelo ChatGPT é apenas um dos vários modelos criados pela empresa nos últimos anos. A seguir, apresentamos alguns dos principais modelos e suas funções:

- GPT-4: Trata-se de um modelo de linguagem de grande escala (LLM) multimodal, que aceita entradas de texto ou imagem e produz texto. Ele é capaz de resolver problemas complexos com maior precisão do que os modelos anteriores da OpenAI.
- DALL-E: Este sistema de IA pode criar imagens e arte realistas a partir de descrições textuais. O DALL-E 3, em particular, suporta a capacidade de gerar novas imagens em tamanhos específicos, com base em sugestões fornecidas.
- TTS: É um modelo de IA que transforma texto em fala com uma sonoridade natural.
- Whisper: O Whisper é um modelo de reconhecimento de fala de uso geral, treinado em um vasto conjunto de dados de áudio diversificado. Ele é um modelo multi-tarefa capaz de realizar reconhecimento de fala em múltiplos idiomas, além de tradução de fala e identificação de idioma.

Acessando a documentação da API

Para acessar a API da documentação, basta clicar [aqui](#).

The screenshot shows the 'Documentation' section of the OpenAI developer platform. On the left, there's a sidebar with navigation links like 'Documentation', 'API reference', 'GET STARTED' (with 'Overview' selected), 'CAPABILITIES' (with 'Text generation' selected), 'ASSISTANTS', and 'GUIDES'. The main content area has a title 'Welcome to the OpenAI developer platform' and a section 'Start with the basics' containing two cards: 'Quickstart tutorial' (purple gradient background) and 'Prompt examples' (red gradient background). Below this are sections for 'Build an assistant' (with 'Introduction' and 'Assistants deep dive' cards) and 'Explore the API' (with 'Text generation', 'Embeddings', 'Image generation', 'Text to speech', 'Prompt engineering', 'Speech to text', 'Fine-tuning', and 'Vision' cards). Each card includes a small icon, a title, and a brief description.

Explorando a API da OpenAI

Capabilities

Na área de [capabilities](#) ficam as principais instruções de como utilizar a API.

The screenshot shows the 'Text generation models' section of the OpenAI documentation. On the left, there's a sidebar with a red box highlighting the 'Text generation' category under 'CAPABILITIES'. The main content area has a heading 'Text generation models' and a paragraph explaining that these models are trained to understand natural language, code, and images. It lists several applications like drafting documents, writing computer code, and analyzing texts. Below this is a section for 'Explore GPT-4 Turbo with Image Inputs' and a button to 'Try out GPT-4 Turbo'. At the bottom, there are links for 'MODEL FAMILIES' and 'API ENDPOINT'.

Modelos

Na [página de modelos](#), você pode ver a descrição completa de todos os modelos disponíveis hoje da OpenAI.

The screenshot shows the 'Models' section of the OpenAI documentation. On the left, there's a sidebar with a red box highlighting the 'Overview' category under 'CAPABILITIES'. The main content area has a heading 'Models' and a sub-section 'Overview' which describes the diverse set of models available. It includes a table comparing different models based on their capabilities. Below the table, it mentions open source models like Point-E, Whisper, Jukebox, and CLIP. There's also a section for 'Continuous model upgrades' with a note about the latest model versions.

MODEL	DESCRIPTION
GPT-4 and GPT-4 Turbo	A set of models that improve on GPT-3.5 and can understand as well as generate natural language or code
GPT-3.5 Turbo	A set of models that improve on GPT-3.5 and can understand as well as generate natural language or code
DALL·E	A model that can generate and edit images given a natural language prompt
TTS	A set of models that can convert text into natural sounding spoken audio
Whisper	A model that can convert audio into text
Embeddings	A set of models that can convert text into a numerical form
Moderation	A fine-tuned model that can detect whether text may be sensitive or unsafe
GPT Base	A set of models without instruction following that can understand as well as generate natural language or code
How we use your data	
Endpoint compatibility	
Tutorials	
Changelog	
Deprecated	A full list of models that have been deprecated along with the suggested replacement

Explorando a API da OpenAI

Exemplos de prompts

Na [página de exemplos](#), você pode verificar alguns prompts simples mas eficientes para diversas tarefas diferentes:

The screenshot shows the 'Prompt examples' section of the OpenAI documentation. At the top, there's a search bar and a category selector. Below, a heading says 'Explore what's possible with some example prompts'. A grid of 12 cards, each with an icon and a brief description, follows:

- Grammar correction**: Convert ungrammatical statements into standard English.
- Summarize for a 2nd grader**: Simplify text to a level appropriate for a second-grade student.
- Parse unstructured data**: Create tables from unstructured text.
- Emoji Translation**: Translate regular text into emoji text.
- Calculate time complexity**: Find the time complexity of a function.
- # Explain code**: Explain a complicated piece of code.
- Keywords**: Extract keywords from a block of text.
- Product name generator**: Generate product names from a description and seed words.
- Python bug fixer**: Find and fix bugs in source code.
- Spreadsheet creator**: Create spreadsheets of various kinds of data.
- # Tweet classifier**: Detect sentiment in a tweet.
- Airport code extractor**: Extract airport codes from text.
- Mood to color**: Turn a text description into a color.
- VR fitness idea generator**: Generate ideas for fitness promoting virtual reality games.

Cookbook

A OpenAI fornece a explicação completa de diversos projetos desenvolvidos pela sua equipe na página de [Cookbook](#).

The screenshot shows the 'Cookbook' section of the OpenAI documentation. At the top, there's a search bar and navigation links. Below, two sections are displayed:

New APIs

Processing and narrating a video with GPT's visual capabilities and the TTS API Kai Chen Nov 6, 2023 COMPLETIONS SPEECH VISION	What's new with DALL-E 3? Will Depue Nov 6, 2023 DALL-E	Assistants API Overview (Python SDK) Ilan Bigio Nov 10, 2023 ASSISTANTS FUNCTIONS
Creating slides with the Assistants API and DALL-E 3 James Hills Dec 8, 2023 ASSISTANTS DALL-E	Using logprobs James Hills, Shyamal Anadkat Dec 20, 2023 COMPLETIONS	Using GPT4 with Vision to tag and caption images Katia Gil Guzman Feb 28, 2024 EMBEDDINGS VISION

Popular

How to call functions with chat models Colin Jarvis, Joe Palermo Jun 13, 2023 COMPLETIONS FUNCTIONS	How to count tokens with Tiktoken Ted Sanders Dec 16, 2022 COMPLETIONS TIKTOKEN	Data preparation and analysis for chat model fine-tuning Michael Wu, Simón Fishman Aug 22, 2023 COMPLETIONS TIKTOKEN
How to stream completions Ted Sanders	Question answering using embeddings-based search Ted Sanders, Mike Heaton	How to format inputs to ChatGPT models Ted Sanders

Explorando a API da OpenAI

Guias

Também está disponível diversos [guias](#) de boas práticas recomendadas ao utilizar os modelos da OpenAI.

The screenshot shows the OpenAI API documentation interface. On the left, there's a sidebar with various sections like 'Documentation', 'API reference', 'Fine-tuning', 'Image generation', 'Vision', 'Text-to-speech', 'Speech-to-text', 'Moderation', 'ASSISTANTS', 'Overview', 'How Assistants work', 'Tools', and 'GUIDES'. The 'GUIDES' section is expanded, and the 'Prompt engineering' guide is selected, highlighted with a red border. The main content area to the right is titled 'Prompt engineering' and contains several sections: 'This guide shares strategies and tactics for getting better results from large language models (sometimes referred to as GPT models) like GPT-4. The methods described here can sometimes be deployed in combination for greater effect. We encourage experimentation to find the methods that work best for you.', 'Some of the examples demonstrated here currently work only with our most capable model, gpt-4. In general, if you find that a model fails at a task and a more capable model is available, it's often worth trying again with the more capable model.', 'You can also explore example prompts which showcase what our models are capable of:', 'Prompt examples' (with a button to 'Explore prompt examples to learn what GPT models can do'), 'Six strategies for getting better results', 'Write clear instructions', 'Provide reference text', and 'Tactics' (a bulleted list: 'Include details in your query to get more relevant answers', 'Ask the model to adopt a persona', 'Use delimiters to clearly indicate distinct parts of the input', 'Specify the steps required to complete a task', 'Provide examples', 'Specify the desired length of the output').

03. Adicionando saldo a sua conta da OpenAI

É importante esclarecermos, desde o início do nosso curso, que será necessário realizar um pequeno investimento para utilizar a API da OpenAI. Quando digo pequeno, refiro-me a um valor acessível, especialmente considerando as capacidades e o poder da API, embora ainda represente um custo.

Recomendo enfaticamente que façam esse investimento. Apesar da existência de modelos gratuitos no mercado, atualmente nenhum se equipara aos oferecidos pela OpenAI. Os modelos que empregaremos são extremamente avançados e requerem uma capacidade computacional imensa para funcionar. Mesmo que tivéssemos acesso ao código-fonte, não conseguiríamos operá-los em nossos computadores pessoais. Ao utilizar a API, não somos nós que processamos os dados, mas sim os servidores da própria OpenAI. É justo, portanto, que a empresa seja compensada por disponibilizar uma quantidade significativa de recursos computacionais para nosso uso. Contribuir financeiramente para a OpenAI significa também apoiar o desenvolvimento contínuo dos modelos de linguagem de grande escala (LLM) e todos os benefícios potenciais que eles trazem.

Há a opção de usar a API de forma gratuita, e abordaremos as limitações dessa modalidade antes de discutirmos como adicionar saldo. Se você optar por iniciar o curso sem adicionar fundos, tudo bem. Contudo, esteja ciente de que, se em algum momento você se deparar com limitações, será necessário adicionar saldo para prosseguir.

Limites de uso gratuito

[Nesta página](#), você pode verificar todas as limitações de uso por tipo de modelo. Os limites são portanto:

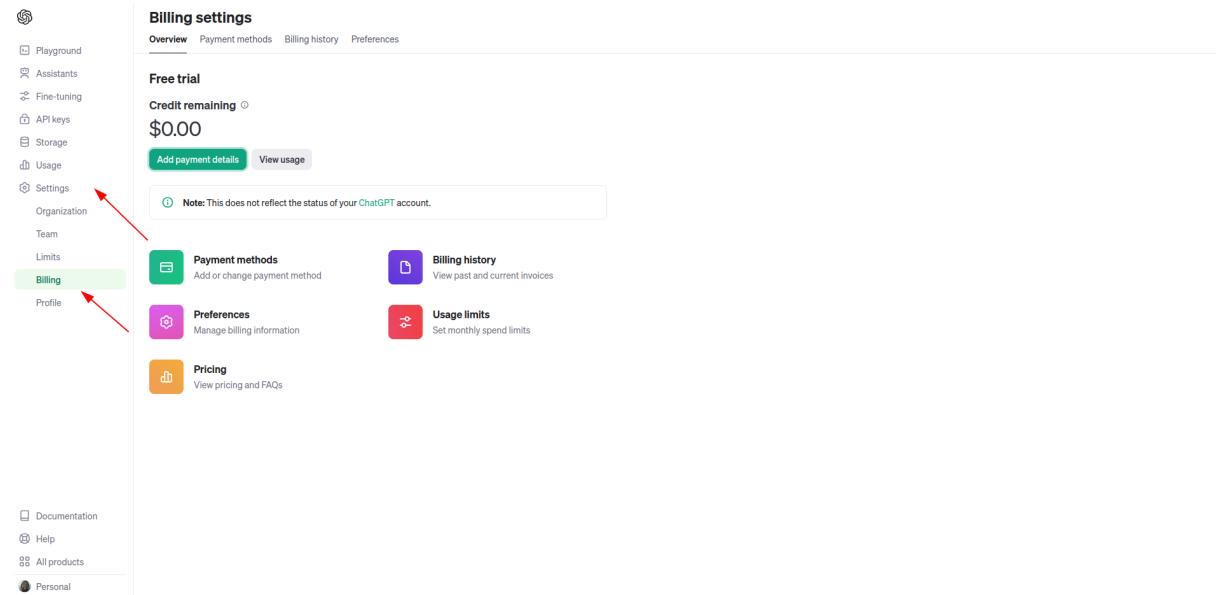
Modelo	RPM	RPD	TPM
gpt-3.5-turbo	3	200	40,000
text-embedding-3-small	3	200	150,000
whisper-1	3	200	-
tts-1	3	200	-
dall-e-2	5 imagem/min	-	-
dall-e-3	1 imagem/min	-	-

Sendo: - RPM: requisições por minuto - RPD: requisições por dia - TPM: tokens por minuto

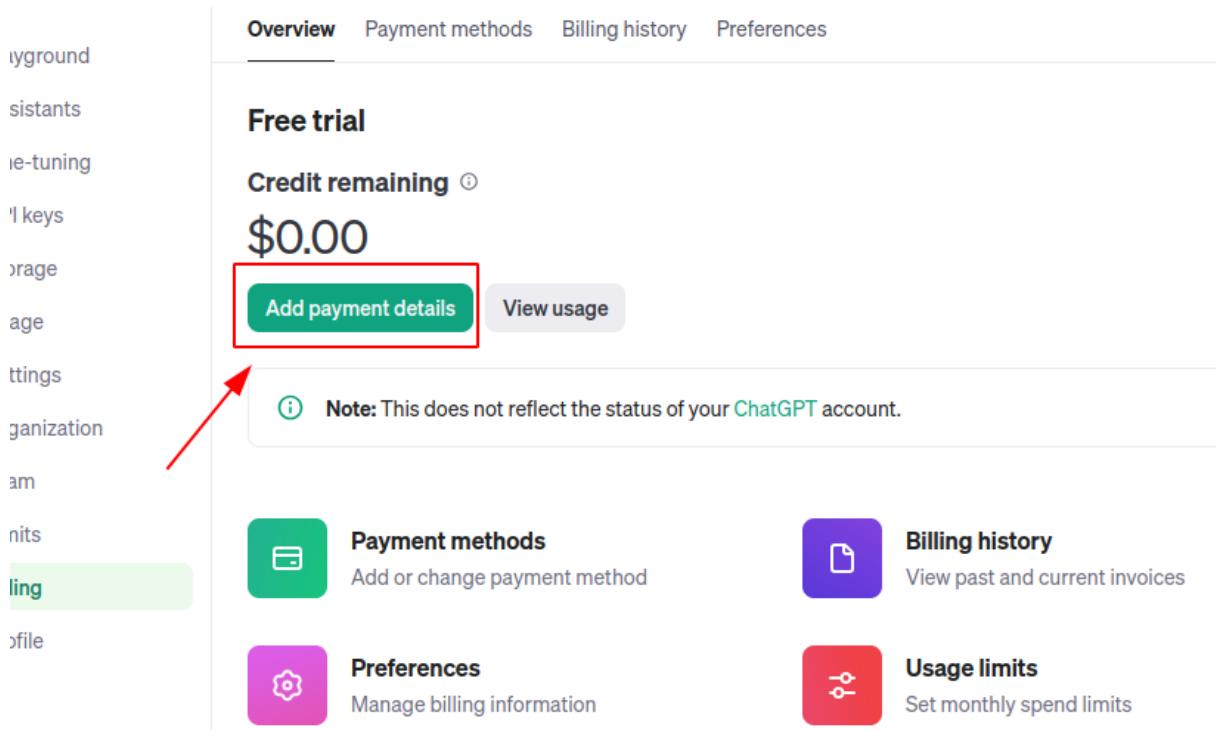
Você pode perceber que modelos mais avançados como o GPT-4 não está disponível a nível gratuito.

Adicionando saldo

Para adicionar saldo, você precisa ir para a [página de billing](#), clicando antes em **Settings**:

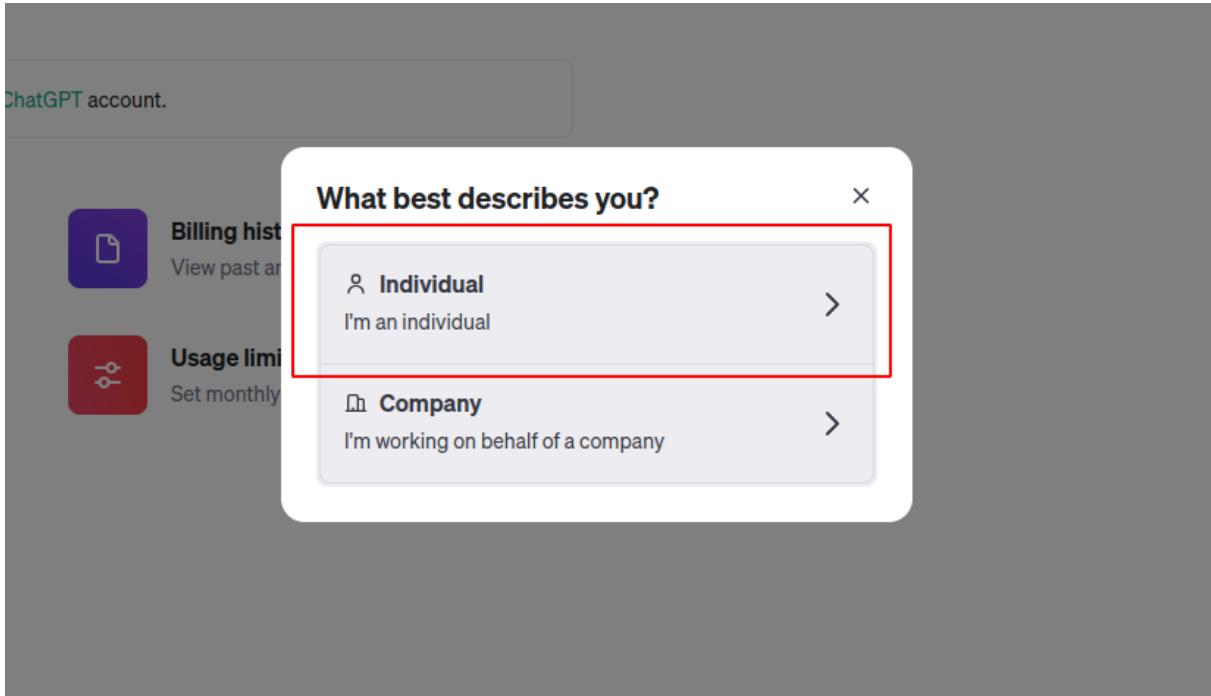


Depois é só clicar em Add Payment details:

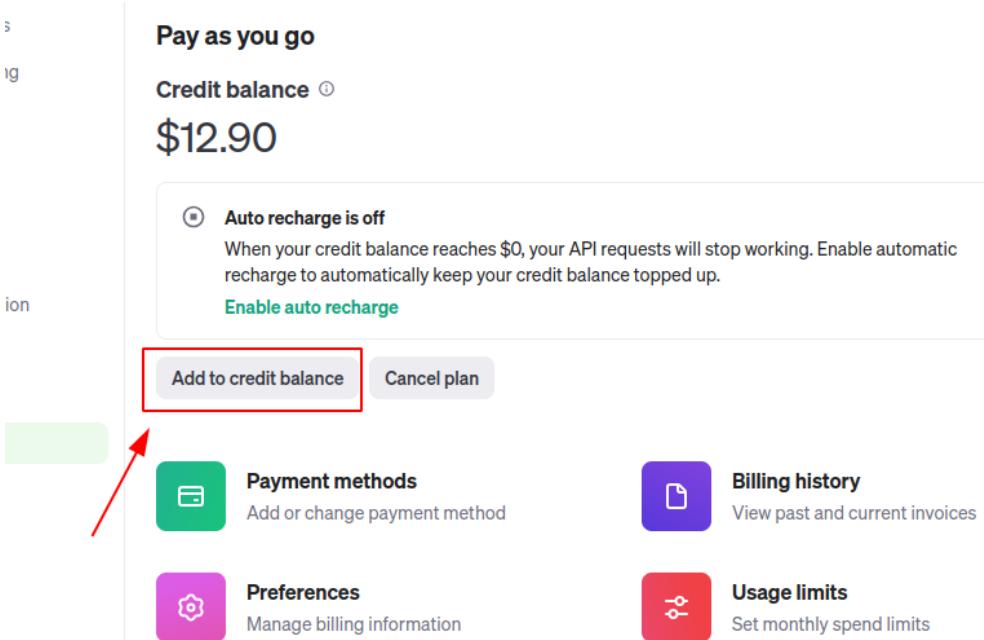


Clicar em Individual, caso for um cartão em nome de uma pessoa física:

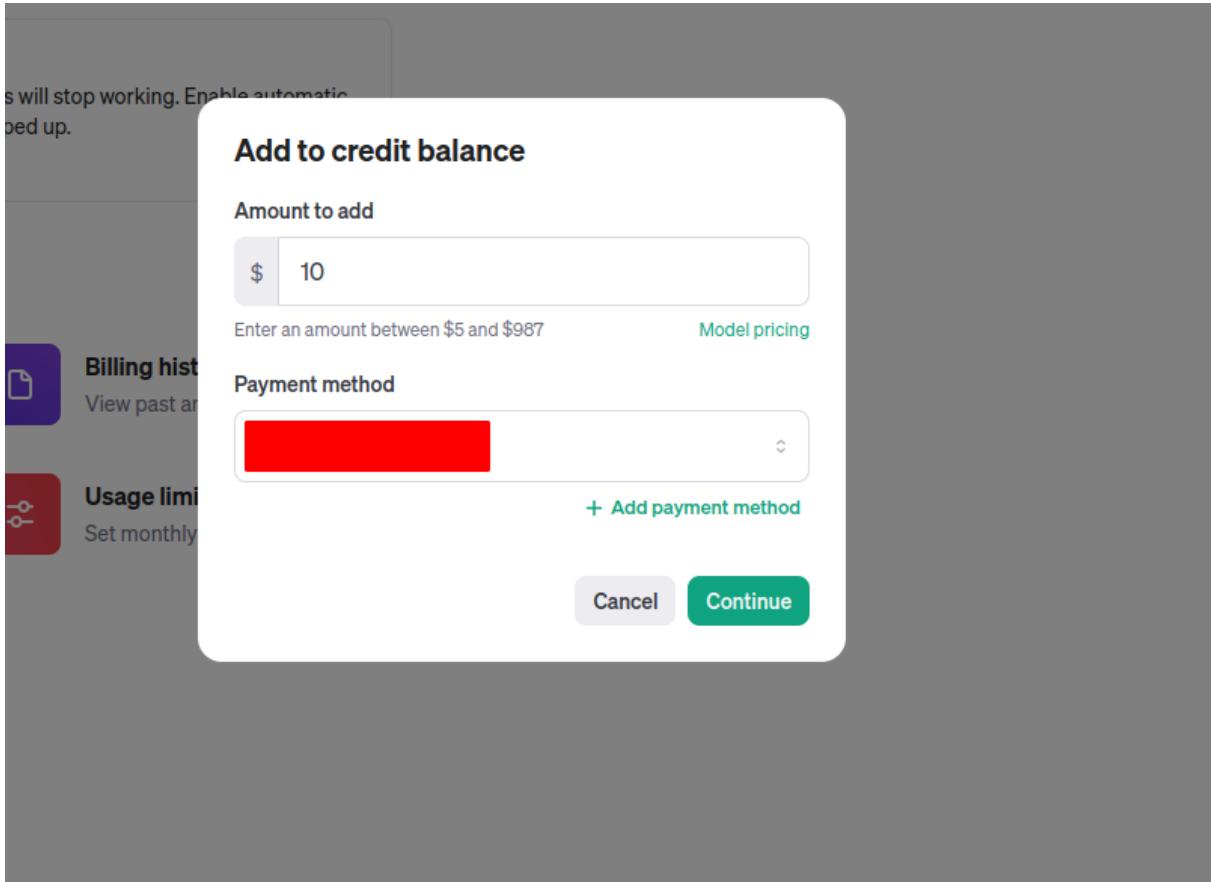
Explorando a API da OpenAI



E aparecerá uma tela para você adicionar suas informações de cartão. Após adicionado, será possível aportar saldo a sua conta, clicando em Add to credit balance:



Selecionando o valor e adicionando:



O menor valor possível é de U\$5, o que equivale a aproximadamente R\$25 na cotação de hoje. Clicando em Continue, adicionamos saldo a nossa conta.

Adicionando limites de gasto mensal

Algo importante é controlar os gastos gerados pela API. A Openai pensou nisto, naturalmente, e adicionou um limitador de custos mensais na aba de [limits](#).

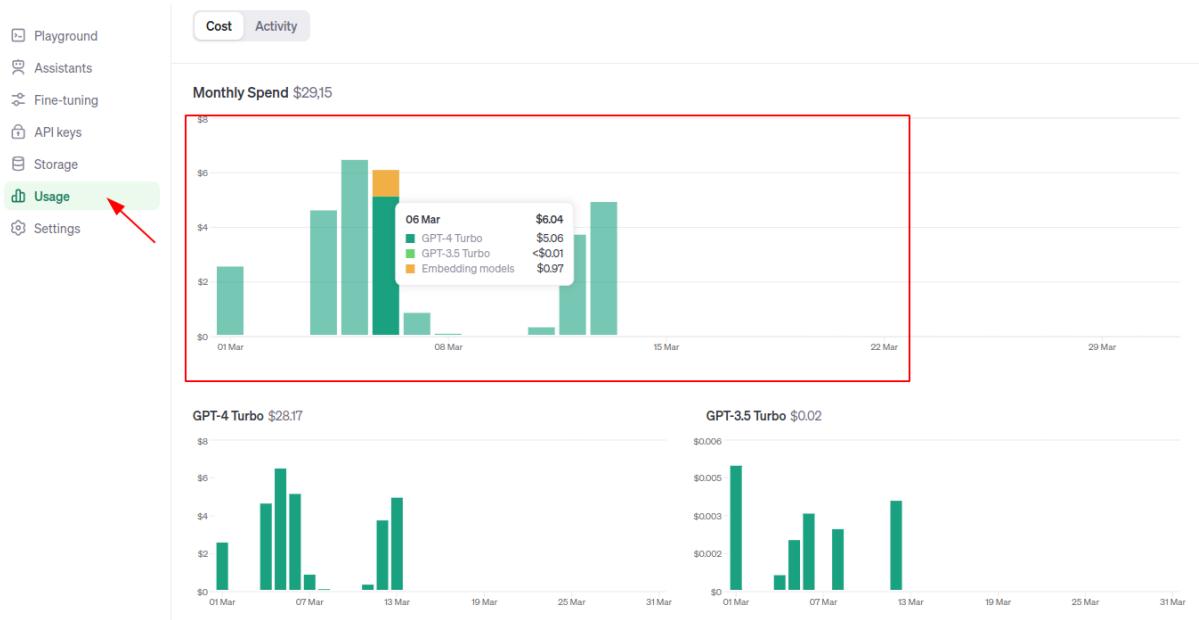
Explorando a API da OpenAI

The screenshot shows the 'Usage limits' section of the OpenAI API settings. On the left sidebar, the 'Limits' option is highlighted with a green box and a red arrow pointing to it. The main area contains a summary of usage limits, a budget input field set to '\$120.00' (which is highlighted with a red box), and an email notification threshold input field set to '\$96.00'. A 'Save' button is at the bottom.

Aqui você pode facilmente configurar os limites de gastos em dólares. Basta digitar um novo valor na caixa e clicar no botão Save.

Verificação de uso

No momento que você criou uma aplicação utilizando as APIs, será importatne monitorar os custos que você está tendo por dia e por modelo diferente. Isso é possível na aba *Usage*.



Passando o mouse sobre o gráfico podemos verificar o custo por cada modelo.

Custos dos modelos

Já sabemos controlar nossos custos, mas é importante entender quais custos são estes. Para isto, vamos a página de [precificação da Openai](#).

Estes são os preços atuais para o modelo GPT-4 Turbo:

Model	Input	Output
gpt-4-0125-preview	\$10.00 / 1M tokens	\$30.00 / 1M tokens
gpt-4-1106-preview	\$10.00 / 1M tokens	\$30.00 / 1M tokens
gpt-4-1106-vision-preview	\$10.00 / 1M tokens	\$30.00 / 1M tokens

É importante lembrar que um token equivale a aproximadamente quatro letras. Como podemos observar, os custos são calculados com base tanto nos tokens de entrada (o prompt) quanto na saída (resposta do modelo).

Os diferentes modelos têm diferentes especificações. No caso dos modelos de imagem, a cobrança é feita por imagem gerada:

Model	Quality	Resolution	Price
DALL-E 3	Standard	1024×1024	\$0.040 / image
	Standard	1024×1792, 1792×1024	\$0.080 / image
DALL-E 3	HD	1024×1024	\$0.080 / image
	HD	1024×1792, 1792×1024	\$0.120 / image
DALL-E 2		1024×1024	\$0.020 / image
		512×512	\$0.018 / image
		256×256	\$0.016 / image

Modelo de áudio para texto como o *Whisper* cobram por minuto transscrito e modelos de texto para áudio cobram por caractere convertido:

Model	Usage
Whisper	\$0.006 / minute (rounded to the nearest second)
TTS	\$15.00 / 1M characters
TTS HD	\$30.00 / 1M characters

04. Inicializando a biblioteca da OpenAI

Agora vamos para a instalação da biblioteca de Python para começar a brincar com nossas APIs.

Instalando a biblioteca

A biblioteca foi desenvolvida pela própria OpenAI e serve como um facilitador para os usuários que desejam acessar os recursos da empresa. Você pode encontrar o repositório da biblioteca [aqui](#). Para instalá-la, você pode usar o pip com o seguinte comando:

```
pip install openai
```

Este curso é feito utilizando a versão 1.14.0 da API. Caso você queira utilizar a mesma versão e garantir a compatibilidade, basta rodar o seguinte:

```
pip install openai==1.14.0
```

E um alerta importante:

O curso não é compatível para versões anteriores a 1.0.0!

Portanto, se você está utilizando uma versão mais antiga, faça a atualização utilizando o seguinte código:

```
pip install openai --upgrade
```

Gerando uma API Key

Para associar as chamadas que você realiza à sua conta da OpenAI, é necessário gerar uma chave de API (API key) e fornecê-la ao inicializar a biblioteca. Para criar essa chave, você deve retornar ao site da OpenAI.

Vamos até a aba de [API Keys](#) e clicamos em *Create new secret key*:

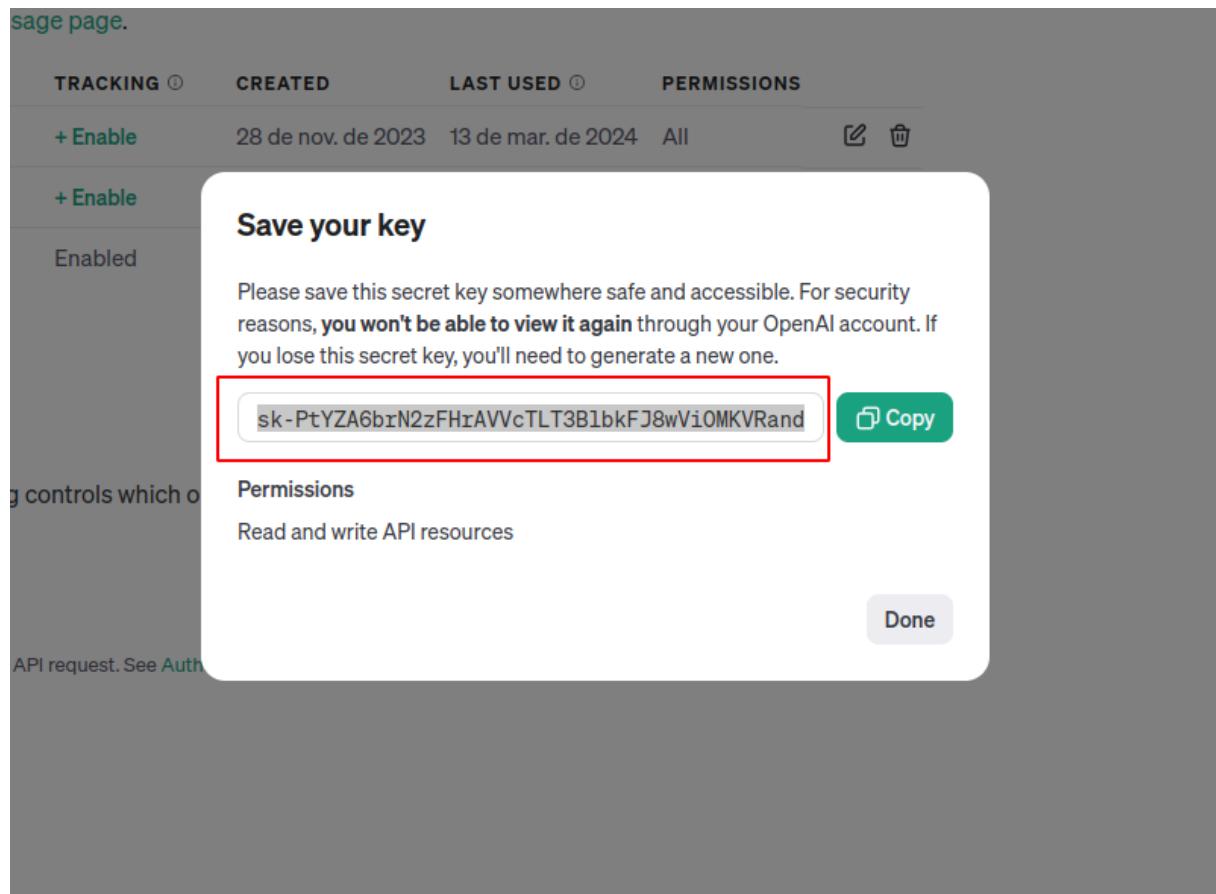
Explorando a API da OpenAI

The screenshot shows the 'API keys' section of the OpenAI dashboard. On the left, a sidebar lists 'Playground', 'Assistants', 'Fine-tuning', 'API keys' (which is highlighted with a red arrow), 'Storage', 'Usage', and 'Settings'. The main area displays two API keys: 'AsimoChat_server' and 'blog_from_transcript'. Both keys have tracking enabled, were created on November 28, 2023, and last used on March 13, 2024. Under 'Permissions', both are set to 'All'. A red box highlights the '+ Create new secret key' button at the bottom of the list.

Adicionamos um nome para identificar a chave e clicamos em *Create secret key*:

A modal dialog titled 'Create new secret key' is displayed over the main API keys list. It contains a 'Name' field with 'chave_teste' typed in, a 'Permissions' section with 'All' selected, and a 'Create secret key' button at the bottom right, which is also highlighted with a red box.

Após a criação da chave, o valor será exibido para você. É essencial que você guarde esse valor de forma segura e não o divulgue, pois ele só aparecerá uma vez. Depois de clicar no botão “done”, não será mais possível verificar o valor da chave, então, certifique-se de salvá-lo imediatamente!



Perfeito, agora temos nossa key, biblioteca instalada e podemos inicializar nosso primeiro cliente.

Inicializando o cliente

O clinete é a classe de comunicação com todos os recursos da API. Para inicializá-lo, será necessário termos em mão nossa api_key e rodar o seguinte comando:

```
import openai  
  
api_key = 'XXXXXXXXXXXXXX'  
client = openai.Client(api_key=api_key)
```

Por questões de segurança, é recomendável não expor nossas chaves de API nos scripts que escrevemos. Para evitar essa exposição, podemos utilizar a biblioteca python-dotenv, que permite ler um arquivo .env e definir as variáveis contidas nele como variáveis de ambiente. Primeiro, criamos nosso arquivo .env com o seguinte conteúdo:

```
OPENAI_API_KEY=XXXXXXXXXXXXXXXXXXXXXX
```

Você deve adicionar no valor em X a sua chave de API. Após isso, vamos instalar a biblioteca python-dotenv:

```
pip install python-dotenv
```

E agora podemos carregar ao nosso script, com o seguinte código:

```
from dotenv import load_dotenv, find_dotenv  
  
_ = load_dotenv(find_dotenv())
```

O código final recomendado para inicializar o cliente é o seguinte:

```
import openai  
from dotenv import load_dotenv, find_dotenv  
_ = load_dotenv(find_dotenv())  
  
client = openai.Client()
```

Observe que, com essa abordagem, não é necessário referenciar a sua API key diretamente no código, pois a biblioteca da OpenAI buscará automaticamente a chave entre as variáveis de ambiente. Se a chave não for encontrada nas variáveis de ambiente, um erro será retornado.

05. Utilizando o ChatGPT para gerar textos

Vamos aprender a gerar textos por meio da API. O objetivo é replicar as funcionalidades disponíveis na interface do ChatGPT, utilizando agora a linguagem Python.

Escrevendo seu primeiro script de geração de texto

Vamos mostrar de forma rápida um script completo e depois vamos analisando linha a linha as minúcias do código:

```
import openai
from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv())

client = openai.Client()

mensagens = [{"role": "user", "content": "O que é uma maçã em até 5 palavras?"}]
resposta = client.chat.completions.create(
    messages=mensagens,
    model='gpt-3.5-turbo-0125',
    max_tokens=1000,
    temperature=0,
)
mensagem_resp = resposta.choices[0].message
print(mensagem_resp.content)
```

Fruta redonda e saborosa.

Poderíamos continuar a conversa com o seguinte script:

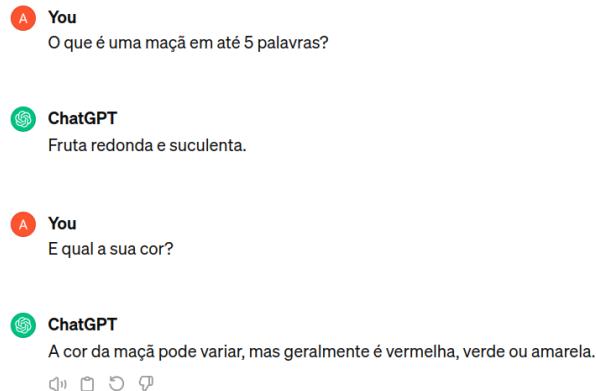
```
mensagens += [mensagem_resp.model_dump(exclude_none=True)]
mensagens += [{"role": "user", "content": "E qual a sua cor?"}]

resposta = client.chat.completions.create(
    messages=mensagens,
    model='gpt-3.5-turbo-0125',
    max_tokens=1000,
    temperature=0,
)
mensagem_resp = resposta.choices[0].message
print(mensagem_resp.content)
```

Vermelha ou verde.

E isso foi equivalente ao seguinte na interface do ChatGPT:

ChatGPT 3.5 ▾



Analisando o script inicial

Parâmetro messages

Para comunicar-se com o modelo, utilizamos o parâmetro `messages`. É necessário fornecer uma lista de dicionários ao método `chat.completions.create` do `client`. Cada dicionário contém duas chaves principais: `content`, que é o conteúdo da mensagem, e `role`, que define o papel de quem está enviando a mensagem. A chave `role` pode ter três valores distintos:

- **system**: A mensagem de sistema é usada para orientar o comportamento do assistente. Por exemplo, é possível alterar a personalidade do assistente ou dar instruções específicas sobre como ele deve agir durante a conversa. Vale ressaltar que a mensagem de sistema é opcional, e a ausência dela pode resultar em um comportamento padrão do modelo, semelhante ao que ocorreria com uma mensagem genérica como “Você é um assistente prestativo”.
- **user**: As mensagens de usuário são as solicitações ou comentários que o assistente deve responder.
- **assistant**: As mensagens do assistente representam as respostas anteriores ou podem ser criadas pelo usuário para exemplificar o comportamento desejado do modelo. Isso é útil, por

exemplo, na técnica de few-shot learning, onde uma sequência de mensagens marcadas como user e assistant serve para demonstrar ao modelo como responder corretamente.

A seguir, um exemplo de como poderia ser estruturado um few-shot learning com mensagens específicas:

```
mensagens = [
    {'role': 'system', 'content': 'Você avalia comentários em positivo ou negativo.'},
    {'role': 'user', 'content': 'Isso é incrível!'},
    {'role': 'assistant', 'content': 'Positivo'},
    {'role': 'user', 'content': 'Isso é ruim!'},
    {'role': 'assistant', 'content': 'Negativo'},
    {'role': 'user', 'content': 'Que programa horrível!'},
]

resposta = client.chat.completions.create(
    messages=mensagens,
    model='gpt-3.5-turbo-0125',
    max_tokens=1000,
    temperature=0,
)

mensagem_resp = resposta.choices[0].message
print(mensagem_resp.content)
```

Negativo

Analisando a classe de resposta do modelo

Podemos observar que a resposta do modelo é sempre retornada no mesmo tipo de dado: a classe ChatCompletion da biblioteca da OpenAI:

```
import openai
from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv())

client = openai.Client()

mensagens = [{'role': 'user', 'content': 'O que é uma maçã em até 5 palavras?'}]
resposta = client.chat.completions.create(
    messages=mensagens,
    model='gpt-3.5-turbo-0125',
    max_tokens=1000,
    temperature=0,
)

print(type(resposta))
```

Ela possui algumas propriedades interessantes:

- `id`: id único da sua chamada e serve para identificá-la

```
print(resposta.id)
```

chatcmpl-92frfZEkVrXD3yCTiEAJNPumPR8L6

- `usage`: permite verificar quantos tokens forma enviados ao modelo e quanto retornaram na resposta. A partir desse valor, podemos calcular o custo da chamada

```
print(resposta.usage)
```

`CompletionUsage(completion_tokens=9, prompt_tokens=21, total_tokens=30)`

- `choices`: é por último, o parâmetro que armazena a resposta do modelo

```
print(resposta.choices)
```

[`Choice(finish_reason='stop', index=0, logprobs=None, message=ChatCompletionMessage(content='Fruta redonda e saborosa.', role='assistant'))`]

Para retornar a mensagem de fato, precisamos retornar o elemento zero do `choices` e depois o parâmetro `message`:

```
mensagem_resp = resposta.choices[0].message
print(mensagem_resp)
```

`ChatCompletionMessage(content='Fruta redonda e saborosa.', role='assistant')`

Analizando a mensagem de resposta

A resposta final é fornecida pela classe `ChatCompletionMessage`. Esta classe tem quatro propriedades: `content`, `role`, `function_calle tool_calls`. Por ora, vamos focar apenas nas duas primeiras propriedades. Os termos `content` e `role` já são familiares, pois correspondem ao mesmo formato que utilizamos para enviar a mensagem inicial. Para incorporar a resposta do modelo à nossa mensagem original, poderíamos proceder de duas maneiras:

```
mensagem_resp = resposta.choices[0].message
mensagens.append({'role': mensagem_resp.role, 'content': mensagem_resp.content})
```

Ou utilizando o método da classe `ChatCompletionMessage` para transformar a resposta em um dicionário:

```
mensagens.append(mensagem_resp.model_dump(exclude_none=True))
```

Demais parâmetros do método chatcompletion

Relembrando, ao chamarmos a função, além do parâmetro *messages*, passamos também os argumentos *model*, *max_tokens*, *temperature*.

model São os modelos de linguagem atuais da openai. A lista você pode acessar diretamente na documentação:

The screenshot shows the 'Models' section of the OpenAI API documentation. On the left, there's a sidebar with links like Overview, Introduction, Quickstart, Models, Overview (which is highlighted with a red arrow), Model updates, GPT-4, GPT-3.5 Turbo, DALL-E, TTS, Whisper, Embeddings, Moderation, GPT Base, and Minha sua user data. The main content area has a title 'Models' and a sub-section 'Overview'. It contains a brief description of the API's model ecosystem and a table comparing different models based on their capabilities. The 'GPT-4 and GPT-4 Turbo' and 'GPT-3.5 Turbo' rows are specifically highlighted with red boxes.

MODEL	DESCRIPTION
GPT-4 and GPT-4 Turbo	A set of models that improve on GPT-3.5 and can understand as well as generate natural language or code
GPT-3.5 Turbo	A set of models that improve on GPT-3.5 and can understand as well as generate natural language or code
DALLE	A model that can generate and edit images given a natural language prompt
TTS	A set of models that can convert text into natural sounding spoken audio
Whisper	A model that can convert audio into text

Alguns modelos disponíveis hoje:

MODEL	CONTEXT WINDOW	TRAINING DATA
gpt-3.5-turbo-0125	16,385 tokens	Up to Sep 2021
gpt-3.5-turbo	16,385 tokens	Up to Sep 2021
gpt-3.5-turbo-1106	16,385 tokens	Up to Sep 2021
gpt-4-0125-preview	128,000 tokens	Up to Dec 2023
gpt-4-turbo-preview	128,000 tokens	Up to Dec 2023
gpt-4-1106-preview	128,000 tokens	Up to Apr 2023
gpt-4-vision-preview	128,000 tokens	Up to Apr 2023
gpt-4-1106-vision-preview	128,000 tokens	Up to Apr 2023
gpt-4	8,192 tokens	Up to Sep 2021

No método *chatcompletion.create*, basta passar uma string com o nome do modelo para utilizá-lo.

max_tokens O parâmetro *max_tokens* representa o tamanho máximo da resposta. Ele serve como uma medida de proteção contra custos excessivos da API, mas também pode resultar em respostas incompletas se o limite for muito baixo.

Explorando a API da OpenAI

```
mensagens += [mensagem_resp.model_dump(exclude_none=True)]
mensagens += [{'role': 'user', 'content': 'E qual a sua cor?'}]

resposta = client.chat.completions.create(
    messages=mensagens,
    model='gpt-3.5-turbo-0125',
    max_tokens=5,
    temperature=0,
)
mensagem_resp = resposta.choices[0].message
print(mensagem_resp.content)
```

Vermelha ou

temperature A “temperatura” é um parâmetro que nos permite controlar o grau de determinismo do modelo. Quanto mais determinístico, maior é a tendência do modelo em escolher o token de maior probabilidade como o próximo na sequência. Um aumento na temperatura resulta em respostas mais aleatórias, o que pode ser interpretado como um aumento na “criatividade” do modelo. Geralmente, o modelo aceita valores de temperatura que variam de 0 a 2.

```
mensagens += [mensagem_resp.model_dump(exclude_none=True)]
mensagens += [{'role': 'user', 'content': 'E qual a sua cor?'}]

resposta = client.chat.completions.create(
    messages=mensagens,
    model='gpt-3.5-turbo-0125',
    max_tokens=100,
    temperature=0,
)
mensagem_resp = resposta.choices[0].message
print(mensagem_resp.content)
```

Vermelha ou verde.

```
mensagens += [mensagem_resp.model_dump(exclude_none=True)]
mensagens += [{'role': 'user', 'content': 'E qual a sua cor?'}]

resposta = client.chat.completions.create(
    messages=mensagens,
    model='gpt-3.5-turbo-0125',
    max_tokens=100,
    temperature=2,
)
mensagem_resp = resposta.choices[0].message
print(mensagem_resp.content)
```

```
Depende da variedadeública/>
Dur.de forma iTreeNode.neighbors(DataResponseStatus      basevoirizeísticaJD_Wind
orpcion Handler_AC_con workingsTransmissionOptions); //Vue Tree.Generic FavoritesP
```

É possível notar que, com a temperatura configurada em 2, o modelo pode gerar respostas imprecisas ou “alucinações”. Portanto, embora valores de temperatura até 2 sejam aceitos, recomenda-se manter o parâmetro de temperatura em até 1 para evitar esse tipo de comportamento.

06. Gerando uma stream de texto

É possível perceber que, ao utilizar a API conforme aprendido na última aula, o modelo gera a resposta completa antes de fornecê-la. Isso difere da experiência ao usar a interface do ChatGPT, onde a resposta parece ser construída de maneira mais interativa.

A You

Crie uma história de dois parágrafos sobre uma viagem a Marte

ChatGPT

Em uma era em que a exploração espacial se tornara rotineira, um grupo de pioneiros embarcou em uma ●

A You

Crie uma história de dois parágrafos sobre uma viagem a Marte

ChatGPT

Em uma era em que a exploração espacial se tornara rotineira, um grupo de pioneiros embarcou em uma jornada épica em direção a Marte. A nave, equipada com tecnologia de ponta, cortava o vazio do espaço, levando consigo sonhos, esperanças e o espírito intrépido dos exploradores. À medida que se aproximavam do Planeta Vermelho, a ansiedade se misturava com a excitação de pisar em um mundo completamente novo, desafiando os limites da imaginação humana. Com corações palpitando de emoção, eles finalmente tocaram a superfície marciana, prontos para desvendar seus segredos e deixar sua marca na história interplanetária. ●

Observamos que a interface do ChatGPT exibe cada novo token gerado, criando uma interação mais dinâmica com o usuário. Felizmente, é possível reproduzir esse mesmo efeito na API utilizando o parâmetro `stream`.

```
import openai
from dotenv import load_dotenv, find_dotenv
```

```
_ = load_dotenv(find_dotenv())

client = openai.Client()

mensagens = [
    {'role': 'user', 'content': 'Crie uma história de dois parágrafos sobre uma viagem a
→ marte'}
]
resposta = client.chat.completions.create(
    messages=mensagens,
    model='gpt-3.5-turbo-0125',
    max_tokens=1000,
    temperature=0,
    stream=True,
)

for stream_resp in resposta:
    print(stream_resp.choices[0].delta.content, end='')
```

Agora, percebemos que, em vez de usarmos a propriedade `message` do objeto `choices`, estamos utilizando o `delta`. Se quiséssemos armazenar a mensagem inteira, poderíamos proceder da seguinte maneira:

```
mensagem_resp = ''
for stream_resp in resposta:
    print(stream_resp.choices[0].delta.content, end='')
    if stream_resp.choices[0].delta.content:
        mensagem_resp += stream_resp.choices[0].delta.content
```

Agora o texto todo ficou armazenado na variável `mensagem_resp`. Atenção ao `if` adicionado:

```
if stream_resp.choices[0].delta.content:
```

Ele é necessário, pois o último `delta` é retornado como `None` e se fossemos concatená-lo a uma string retornaríamos um erro.

07. DESAFIO - Criando um Chatbot em Python

Fica de desafio para você criar o seu primeiro chatbot utilizando a API da openai.

```
(.venv) python 07_desafio.py
Bem-vindo ao ChatBot da Asimov. Digite sua mensagem abaixo!
User: Olá, modelo! Como você vai?
Assistant: Olá! Estou bem, obrigada. Como posso ajudar você hoje?
User: Você tem um nome?
Assistant: Não tenho um nome específico, mas você pode me chamar de assistente virtual. Com o posso ajudar você?
User: Mas se você tivesse um nome humano, qual seria?
Assistant: Que tal me chamar de Sophia? É um nome bonito e fácil de lembrar. O que acha?
User: Achei ótimo, posso passar a te chamar assim?
Assistant: Claro, fico feliz que tenha gostado! Pode me chamar de Sophia sempre que precisar de ajuda. Estou aqui para ajudar você. O que mais posso fazer por você hoje?
User: 
```

Fica o desafio para você tentar resolver. Caso não esteja com vontade, esta é a minha solução:

```
import openai
from dotenv import load_dotenv, find_dotenv

_ = load_dotenv(find_dotenv())
client = openai.Client()

def retorna_resposta_modelo(
    mensagens
):
    resposta = client.chat.completions.create(
        messages=mensagens,
        model='gpt-3.5-turbo-0125',
        max_tokens=1000,
        temperature=0,
        stream=True,
    )

    mensagem_resp = ''
    print('Assistant: ', end='')
    for stream_resp in resposta:
        if stream_resp.choices[0].delta.content:
            print(stream_resp.choices[0].delta.content, end=' ')
            mensagem_resp += stream_resp.choices[0].delta.content
    print()

    mensagens.append(
        {'role': 'assistant', 'content': mensagem_resp}
    )

    return mensagens

def main():
    mensagens = []
    print('Bem-vindo ao ChatBot da Asimov. Digite sua mensagem abaixo!')
```

```
while True:
    user_mensagem = input('User: ')
    mensagens.append(
        {'role': 'user', 'content': user_mensagem}
    )
    mensagens = retorna_resposta_modelo(mensagens)

if __name__ == '__main__':
    main()
```

08. Adicionando funções e ferramentas externas

Uma das limitações de um modelo de linguagem é a sua capacidade de acessar informações atualizadas. Isso ocorre porque o modelo é treinado com dados históricos e, após ser colocado em operação, não continua seu treinamento. Por exemplo, o GPT-4 tem informações atualizadas até dezembro de 2023, e o GPT-3 até setembro de 2021. No entanto, não há motivo para preocupação, pois existe uma solução incrível para obter respostas atuais mesmo sem esses dados na base de conhecimento do modelo: o acesso a ferramentas externas.

Os modelos mais recentes têm a funcionalidade de chamada de funções (function calling). Essas funções podem coletar informações atualizadas para serem processadas pelo modelo ou até mesmo permitir que o próprio modelo execute ações, como enviar e-mails, adicionar valores a uma base de dados, entre outras possibilidades.

A seguir, vamos apresentar um exemplo prático dessa funcionalidade e, depois, analisaremos cada etapa detalhadamente:

```
import json

import openai
from dotenv import load_dotenv, find_dotenv

_ = load_dotenv(find_dotenv())

client = openai.Client()

def obter_temperatura_atual(local, unidade="celsius"):
    if "são paulo" in local.lower():
        return json.dumps(
            {"local": "São Paulo", "temperatura": "32", "unidade": unidade}
        )
    elif "porto alegre" in local.lower():
        return json.dumps(
            {"local": "Porto Alegre", "temperatura": "25", "unidade": unidade}
        )
    elif "rio de janeiro" in local.lower():
        return json.dumps(
            {"local": "Rio de Janeiro", "temperatura": "35", "unidade": unidade}
        )
    else:
        return json.dumps(
            {"local": local, "temperatura": "unknown"}
        )

tools = [
{
    "type": "function",
    "function": {
```

```
"name": "obter_temperatura_atual",
"description": "Obtém a temperatura atual em uma dada cidade",
"parameters": {
    "type": "object",
    "properties": {
        "local": {
            "type": "string",
            "description": "O nome da cidade. Ex: São Paulo",
        },
        "unidade": {
            "type": "string",
            "enum": ["celsius", "fahrenheit"]
        },
        "required": ["local"],
    },
},
],
}

funcoes_disponiveis = {
    "obter_temperatura_atual": obter_temperatura_atual,
}

mensagens = [
    {"role": "user",
     "content": "Qual é a temperatura em São Paulo e Porto Alegre?"}
]

resposta = client.chat.completions.create(
    model="gpt-3.5-turbo-0125",
    messages=mensagens,
    tools=tools,
    tool_choice="auto",
)
mensagem_resp = resposta.choices[0].message
tool_calls = mensagem_resp.tool_calls

if tool_calls:
    mensagens.append(mensagem_resp)
    for tool_call in tool_calls:
        function_name = tool_call.function.name
        function_to_call = funcoes_disponiveis.get(function_name)
        function_args = json.loads(tool_call.function.arguments)
        function_response = function_to_call(
            local=function_args.get("local"),
            unidade=function_args.get("unidade"),
        )
        mensagens.append(
            {
                "tool_call_id": tool_call.id,
                "role": "tool",
                "name": function_name,
                "content": function_response,
```

```
        }
    )
segunda_resposta = client.chat.completions.create(
    model="gpt-3.5-turbo-0125",
    messages=mensagens,
)

mensagem_resp = segunda_resposta.choices[0].message
print(mensagem_resp.content)
```

A temperatura em São Paulo é de 32°C e em Porto Alegre é de 25°C.

É um código grande como vocês podem perceber, mas vamos explicá-lo em detalhes agora.

Definição de funções externas

Iniciamos definindo as funções que o modelo poderá utilizar para extrair informações. No nosso exemplo, criamos uma função para informar a temperatura atual de determinadas localidades. Por se tratar apenas de um exemplo, optamos por manter as temperaturas como valores fixos. No entanto, poderíamos facilmente conectar essa função a uma API de meteorologia para obter dados em tempo real.

```
def obter_temperatura_atual(local, unidade="celsius"):
    if "são paulo" in local.lower():
        return json.dumps(
            {"local": "São Paulo", "temperatura": "32", "unidade": unidade}
        )
    elif "porto alegre" in local.lower():
        return json.dumps(
            {"local": "Porto Alegre", "temperatura": "25", "unidade": unidade}
        )
    elif "rio de janeiro" in local.lower():
        return json.dumps(
            {"local": "Rio de Janeiro", "temperatura": "35", "unidade": unidade}
        )
    else:
        return json.dumps(
            {"local": local, "temperatura": "unknown"}
        )
```

Para garantir que a função retorne uma string, permitindo assim o processamento subsequente da informação pelo modelo, convertemos a saída em formato JSON. Isso é feito utilizando a função `json.dumps`, uma vez que o JSON é um formato de texto amplamente utilizado em APIs para a troca de dados.

Adicionando especificações das ferramentas

Após criar nossa função, é importante informar ao ChatGPT sobre as ferramentas disponíveis para que ele possa utilizá-las. Para isso, montamos um dicionário que descreve as funções. Esse dicionário deve incluir as seguintes chaves:

- **type**: Define o tipo da ferramenta, que neste caso é **function**.
- **function**: Contém a descrição detalhada da função. A chave **function** também é um dicionário e deve conter as seguintes informações:
 - **name**: O nome da função que foi adicionada.
 - **description**: Uma explicação clara do propósito da função, para que o modelo entenda sua utilidade.
 - **parameters**: Os argumentos que a função aceita. Por fim, a chave **parameters** é igualmente um dicionário e deve ter as seguintes chaves:
 - * **type**: Deve ser mantido como ‘object’.
 - * **properties**: Cada chave dentro de **properties** representa um argumento da função, e cada um é definido por:
 - **type**: O tipo de dado do argumento (por exemplo, string, int, etc.).
 - **description**: Uma breve descrição do argumento.
 - **enum**: Uma lista de valores pré-definidos entre os quais o argumento deve ser selecionado, se aplicável.

Com isso, temos uma estrutura organizada que permite ao ChatGPT identificar e utilizar as funções que desenvolvemos.

```
tools = [
{
    "type": "function",
    "function": {
        "name": "obter_temperatura_atual",
        "description": "Obtém a temperatura atual em uma dada cidade",
        "parameters": {
            "type": "object",
            "properties": {
                "local": {
                    "type": "string",
                    "description": "O nome da cidade. Ex: São Paulo",
                },
                "unidade": {
                    "type": "string",
                    "enum": ["celsius", "fahrenheit"]
                },
            },
            "required": ["local"],
        },
    }
},
```

```
    },
}
]
```

É importante observar que utilizamos uma lista de dicionários para descrever as ferramentas disponíveis. Se desejarmos incluir mais de uma função, basta adicionar um novo dicionário à lista, detalhando a nova função. Cada dicionário deve conter a descrição completa da função que se pretende disponibilizar, seguindo o mesmo formato estrutural para garantir a consistência e a compreensão adequada pelo modelo.

Chamando *chat completion* com ferramentas externas

Para informar o modelo sobre as novas funções disponíveis, devemos passar a lista de ferramentas que construímos anteriormente como argumento no parâmetro tools. Isso permite que o modelo reconheça e interaja com as funções que adicionamos, expandindo suas capacidades de processamento e resposta.

```
mensagens = [
    {"role": "user",
     "content": "Qual é a temperatura em São Paulo e Porto Alegre?"}
]

resposta = client.chat.completions.create(
    model="gpt-3.5-turbo-0125",
    messages=mensagens,
    tools=tools,
    tool_choice="auto",
)
```

O parâmetro tool_choice possui o valor “auto” como padrão, o que permite ao modelo decidir automaticamente qual ferramenta utilizar. No entanto, se quisermos direcionar o modelo para usar uma função específica que criamos, podemos alterar o valor padrão de tool_choice para o nome da nossa função. Isso forçará o modelo a utilizar a ferramenta que especificamos.

Entendendo o parâmetro `tool_calls` da resposta do modelo

Para o chamado que realizamos, a resposta do modelo foi:

```
mensagem_resp = resposta.choices[0].message
print(mensagem_resp)
```

```
ChatCompletionMessage(content=None, role='assistant', function_call=None, tool_c
```

A primeira coisa a notar é que o `content` da mensagem retornou `None`, indicando que o modelo não forneceu uma resposta em forma de texto. Outro aspecto relevante é que o parâmetro `tool_calls` agora contém valores, o que sinaliza que o modelo está solicitando a resposta de funções externas para fornecer uma resposta apropriada. Ao analisar o `tool_calls` com mais detalhes, podemos entender melhor como o modelo indica a necessidade de interação com as ferramentas externas.

```
tool_calls = mensagem_resp.tool_calls
print(mensagem_resp)
```

```
[ChatCompletionMessageToolCall(id='call_HF6eGifb06kNpNX3zRuEZyxe', function=Func
```

Podemos notar que há uma lista contendo dois valores, e cada um é representado pela classe `ChatCompletionMessageToolCall`. Ao examinar o primeiro valor, identificamos que o modelo está solicitando a execução da função `obter_temperatura_atual` com os parâmetros `local` definido como “São Paulo” e `unidade` como “celsius”. Isso indica que o modelo requer os dados fornecidos por essa função externa para prosseguir com a resposta adequada.

```
print(tool_calls[0])
```

```
ChatCompletionMessageToolCall(id='call_FYUQWKtHj5k1kpqJy8ru1grb', function=Func
```

E o segundo argumento é a mesma coisa,, mas com o parâmetro `local=“Porto Alegre”`:

```
print(tool_calls[1])
```

```
ChatCompletionMessageToolCall(id='call_FYUQWKtHj5k1kpqJy8ru1grb', function=Func
```

Rodando as funções externas

Agora basta a nós rodar as funções externas e passar novamente para o modelo o resultado:

```
funcoes_disponiveis = {
    "obter_temperatura_atual": obter_temperatura_atual,
}

if tool_calls:
    mensagens.append(mensagem_resp)
    for tool_call in tool_calls:
        function_name = tool_call.function.name
        function_to_call = funcoes_disponiveis[function_name]
        function_args = json.loads(tool_call.function.arguments)
        function_response = function_to_call(
            local=function_args.get("local"),
            unidade=function_args.get("unidade"),
        )
```

```
mensagens.append(  
    {  
        "tool_call_id": tool_call.id,  
        "role": "tool",  
        "name": function_name,  
        "content": function_response,  
    }  
)  
segunda_resposta = client.chat.completions.create(  
    model="gpt-3.5-turbo-0125",  
    messages=mensagens,  
)
```

Primeiro fazemos um if e adicionamos a resposta anterior às mensagens trocadas entre modelo e usuário:

```
if tool_calls:  
    mensagens.append(mensagem_resp)
```

Depois fazemos um for para cada função diferente que deve ser rodada:

```
for tool_call in tool_calls:
```

Utilizamos o dicionário **funcões_disponíveis** como um facilitador para referenciar as funções externas:

```
funcoes_disponiveis = {  
    "obter_temperatura_atual": obter_temperatura_atual,  
}  
  
for tool_call in tool_calls:  
    function_name = tool_call.function.name  
    function_to_call = funcoes_disponiveis[function_name]
```

Como podem observar, os argumentos vem em string no formato json. Para transformá-lo em um dicionário de Python basta realizar o seguinte:

```
function_args = json.loads(tool_call.function.arguments)
```

E pegamos a resposta das funções rodadas:

```
function_response = function_to_call(  
    local=function_args.get("local"),  
    unidade=function_args.get("unidade"),  
)
```

E adicionamos as mensagens que serão enviadas ao modelo:

```
mensagens.append(  
    {  
        "tool_call_id": tool_call.id,  
        "role": "tool",  
        "name": function_name,
```

```
        "content": function_response,  
    }
```

Atenção para a formatação da mensagem! Devemos adicionar o `tool_call_id`, como `role` devemos passar o valor “`tool`”, devemos passar a chave `name` com o nome da função e o `content` com o resultado da função.

E por fim é só rodar o modelo:

```
segunda_resposta = client.chat.completions.create(  
    model="gpt-3.5-turbo-0125",  
    messages=mensagens,  
)
```

Analizando a lista de mensagens final

Vamos analisar as mensagens que foram passadas para o modelo na última chamada:

```
for mensagem in mensagens:  
    print(mensagem, end='\n\n')  
  
{'role': 'user', 'content': 'Qual é a temperatura em São Paulo e Porto Alegre?'},  
  
ChatCompletionMessage(content=None, role='assistant', function_call=None, tool_call_id=None),  
  
{'tool_call_id': 'call_ZAk073PtjNhZieyXorJgdKWP', 'role': 'tool', 'name': 'obter_temperatura'},  
  
{'tool_call_id': 'call_HNAL3thWncnSzh2MYSjhIkN5', 'role': 'tool', 'name': 'obter_temperatura'}
```

Observamos que o total de mensagens é quatro. A primeira mensagem corresponde à nossa pergunta inicial. A segunda é a resposta do modelo, na qual ele solicita informações às ferramentas externas. A terceira e a quarta mensagens são as respostas fornecidas pela ferramenta externa em atendimento aos chamados feitos pelo modelo. Com essas informações em mãos, o modelo foi capaz de nos fornecer a resposta correta!

```
mensagem_resp = segunda_resposta.choices[0].message  
print(mensagem_resp.content)
```

A temperatura em São Paulo é de 32°C e em Porto Alegre é de 25°C.

09. DESAFIO - ChatBot Finanças

O desafio agora é criar um chatbot utilizando a API da openai que tenha acessos a dados do mercado financeiro. Para isso, você pode utilizar a api do yahoo finance:

```
pip install yfinance
```

O seguinte código retorna a cotação histórica para um ativo específico:

```
ticker = 'PETR4'  
ticker_obj = yf.Ticker(f'{ticker}.SA')  
hist = ticker_obj.history(period='1mo')  
print(hist)
```

O parâmetro period pode receber os seguintes valores:

```
["1d", "5d", "1mo", "3mo", "6mo", "1y", "2y", "5y", "10y", "ytd", "max"]
```

- d: dia
- mo: mês
- y: ano
- ytd: desde o início do ano
- max: período máximo

Fica o desafio para você tentar resolver. Caso não esteja com vontade, esta é a minha solução:

```
import json  
import yfinance as yf  
  
import openai  
from dotenv import load_dotenv, find_dotenv  
  
_ = load_dotenv(find_dotenv())  
  
client = openai.Client()  
  
# DEFINE FUNCOES  
def retorna_cotacao_historica(ticker, periodo):  
    ticker_obj = yf.Ticker(f'{ticker}.SA')  
    hist = ticker_obj.history(period=periodo)  
    if len(hist) > 30:  
        slice_size = int(len(hist) / 30)  
        hist = hist.iloc[::slice_size][:-1]  
        hist.index = hist.index.strftime('%m-%d-%Y')  
    return hist['Close'].to_json()  
  
tools = [  
{
```

```
"type": "function",
"function": {
    "name": "retorna_cotacao_historica",
    "description": "Retorna a cotação diária histórica para \
                    uma ação da bovespa",
    "parameters": {
        "type": "object",
        "properties": {
            "ticker": {
                "type": "string",
                "description": "O ticker da ação. Exemplo: 'PETR4' \
                                para petrobras",
            },
            "periodo": {
                "type": "string",
                "description": "O período que será retornado de dados, \
                                sendo '1mo' equivalente a um mês, \
                                '1d' a 1 dia e '1y' a um ano",
                "enum": [
                    "1d", "5d", "1mo", "6mo", "1y", "5y", "10y", "ytd", "max"
                ],
            },
            "required": ["ticker", "periodo"],
        },
    },
},
]

funcoes_disponiveis = {
    "retorna_cotacao_historica": retorna_cotacao_historica,
}

def retorna_resposta_modelo(mensagens):
    resposta = client.chat.completions.create(
        model="gpt-3.5-turbo-0125",
        messages=mensagens,
        tools=tools,
        tool_choice="auto",
    )

    mensagem_resp = resposta.choices[0].message
    mensagens.append(mensagem_resp)

    tool_calls = mensagem_resp.tool_calls
    if tool_calls:
        for tool_call in tool_calls:
            function_name = tool_call.function.name
            function_to_call = funcoes_disponiveis[function_name]
            function_args = json.loads(tool_call.function.arguments)
            function_response = function_to_call(
                ticker=function_args.get("ticker"),
                periodo=function_args.get("periodo"),
            )
            mensagens.append(
```

```
        {
            "tool_call_id": tool_call.id,
            "role": "tool",
            "name": function_name,
            "content": function_response,
        }
    )
segunda_resposta = client.chat.completions.create(
    model="gpt-3.5-turbo-0125",
    messages=mensagens,
)

mensagens.append(segunda_resposta.choices[0].message)
print(mensagens[-1].content)
return mensagens

def main():
    mensagens = []
    print('Bem-vindo ao ChatBot Financeiro da Asimov. \
          Digite sua mensagem abaixo!')
    while True:
        user_mensagem = input('User: ')
        mensagens.append(
            {'role': 'user', 'content': user_mensagem})
    mensagens = retorna_resposta_modelo(mensagens)

if __name__ == '__main__':
    main()
```

11. Fine-Tuning - otimizando um modelo

O que é Fine-Tuning?

O Fine-Tuning em inteligência artificial representa uma técnica de aprendizado por transferência. Essa abordagem consiste em tomar um modelo pré-treinado, o qual foi desenvolvido com base em um vasto conjunto de dados para uma tarefa ampla – a exemplo dos modelos GPT – e realizar ajustes sutis em seus parâmetros internos. O propósito é aprimorar o desempenho do modelo para uma nova tarefa correlata, evitando a necessidade de iniciar o processo de treinamento desde o início.

O Fine-Tuning aprimora o desempenho do prompt *few-shot* ao treinar o modelo com um número de exemplos significativamente maior do que aquele que pode ser acomodado no prompt. Isso possibilita a obtenção de resultados superiores em uma ampla gama de tarefas. Após a otimização de um modelo com Fine-Tuning, torna-se desnecessário fornecer uma quantidade extensa de exemplos no prompt. Essa eficiência resulta em economia de custos e permite realizar solicitações com menor latência.

Quando utilizar Fine-Tuning?

O processo de Fine-tuning em modelos de geração de texto pode aprimorá-los para aplicações específicas, contudo, exige um investimento criterioso de tempo e recursos. Antes de recorrer ao Fine-tuning, é recomendável tentar alcançar resultados satisfatórios por meio de engenharia de prompts, encadeamento de prompts (que consiste em dividir tarefas complexas em múltiplos prompts) e chamada de função. As principais razões para essa abordagem são:

- Muitas tarefas podem parecer desafiadoras para os modelos inicialmente, mas é possível melhorar significativamente os resultados com a escolha adequada de prompts, tornando o Fine-tuning desnecessário.
- O processo de iteração com prompts e outras estratégias permite um ciclo de feedback muito mais ágil do que o Fine-tuning, que demanda a criação de conjuntos de dados e a realização de processos de treinamento.
- Mesmo nos casos em que o Fine-tuning se faz necessário, o trabalho preliminar com engenharia de prompts não é perdido. Geralmente, os melhores resultados são obtidos ao utilizar um prompt bem elaborado nos dados de Fine-tuning, ou ao combinar o encadeamento de prompts e o uso de ferramentas com o Fine-tuning.

Nosso curso de Engenharia de Prompts oferece uma visão abrangente de algumas das estratégias e táticas mais eficientes para melhorar o desempenho dos modelos sem a necessidade de Fine-tuning.

Usos comuns

Alguns casos de uso comuns nos quais o fine-tuning pode aprimorar os resultados incluem:

- Definir o estilo, tom, formato ou outros aspectos qualitativos.
- Aumentar a confiabilidade na geração de uma saída desejada.
- Corrigir falhas ao seguir instruções complexas.
- Lidar com muitos casos atípicos de maneiras específicas.
- Aprender uma nova habilidade ou realizar uma tarefa difícil de articular em uma instrução.

Uma maneira de abordar esses casos é quando é mais eficaz “mostrar, não dizer”.

Criando um modelo com Fine-Tuning em Python

Definição do problema

Nosso objetivo com o exemplo a seguir é formatar as respostas. Desejamos que o modelo forneça sempre sua resposta no formato JSON, com as seguintes chaves: resposta, categoria e fonte. Eis a definição das chaves:

- fonte: o valor deve ser sempre AsimoBot
- resposta: a resposta para a pergunta em um parágrafo contendo até 20 palavras
- categoria: a categoria da pergunta, a qual deve pertencer a uma das seguintes categorias: física, matemática, língua portuguesa ou outras.

Um prompt equivalente que geraria essa resposta seria o seguinte:

```
system_mes = '''  
Responda as perguntas em um parágrafo de até 20 palavras. Categorize as respostas no seguintes  
conteúdos: física, matemática, língua portuguesa ou outros.  
Retorne a resposta em um formato json, com as keys:  
fonte: valor deve ser sempre AsimoBot  
resposta: a resposta para a pergunta  
categoria: a categoria da pergunta  
'''
```

Preparação de dados

A ideia por trás do fine-tuning é fornecer mais exemplos ao modelo, permitindo que ele seja re-treinado com base nesses exemplos e, assim, retorne respostas mais alinhadas com nossas expectativas. Portanto, é essencial gerar dados contendo esses exemplos. A OpenAI requer arquivos de texto no formato JSONL para realizar o treinamento dos modelos. Abaixo, segue um exemplo de dado na formatação necessária:

```
{"messages": [{"role": "user", "content": "instrução_1"}, {"role": "assistant", "messages": [{"role": "user", "content": "instrução_2"}, {"role": "assistant", {"messages": [{"role": "user", "content": "instrução_3"}, {"role": "assistant", {"messages": [{"role": "user", "content": "instrução_4"}, {"role": "assistant",
```

Como você pode perceber, o arquivo consiste em uma lista de mensagens exemplos no mesmo formato que utilizamos na comunicação com o modelo, sempre com as chaves “role” e “content”.

No seguinte código, transformamos um arquivo JSON que está no seguinte formato:

```
[  
  {  
    "pergunta": "O que é um substantivo?",  
    "resposta": "Substantivo é a classe gramatical de palavras que nomeiam seres, objetos,  
    ↳ fenômenos, lugares, qualidades e ações.",  
    "categoria": "Língua Portuguesa"  
  },  
  {  
    "pergunta": "O que é inércia?",  
    "resposta": "Inércia é a tendência de um corpo manter seu estado de repouso ou  
    ↳ movimento uniforme, a menos que forças atuem sobre ele.",  
    "categoria": "Física"  
  }  
]
```

Para um arquivo JSONL:

```
import json  
  
# === Criando o arquivo jsonl para ser enviado a OpenAI  
  
arquivo = 'chatbot_respostas'  
with open(f'{arquivo}.json') as f:  
    conversa_json = json.load(f)  
  
with open(f'{arquivo}.jsonl', 'w') as outfile:  
    for entry in conversa_json:  
        resposta = {  
            'resposta': entry['resposta'],  
            'categoria': entry['categoria'],  
            'fonte': 'AsimoBot'  
        }  
        entry = {"messages": [  
            {"role": "user", "content": entry['pergunta']},  
            {"role": "assistant", "content": json.dumps(resposta, ensure_ascii=False,  
        ↳ indent=2)}]  
        }  
        json.dump(entry, outfile, ensure_ascii=False)  
        outfile.write('\n')
```

Criando modelo com Fine-Tuning

Dado que já criamos nosso modelo, podemos criar um job na OpenAI para criar um novo modelo:

```
import openai
from dotenv import load_dotenv, find_dotenv

_ = load_dotenv(find_dotenv())
client = openai.Client()

arquivo = 'chatbot_respostas'

file = client.files.create(
    file=open(f'{arquivo}.jsonl', "rb"),
    purpose="fine-tune"
)

client.fine_tuning.jobs.create(
    training_file=file.id,
    model="gpt-3.5-turbo"
)
```

Você pode observar que inicialmente um novo arquivo com os exemplos é enviado utilizando o método `files.create`. Em seguida, criamos um job para iniciar o processo com o método `fine_tuning.jobs.create`. É possível notar que passamos o ID do arquivo recém-adicionado e selecionamos o modelo `gpt-3.5-turbo` para otimização.

Verificando status do novo modelo

Você pode utilizar o seguinte comando para verificar os processos de otimização que estão sendo rodados:

```
print(client.fine_tuning.jobs.list(limit=10))
```

E você pode verificar o estado de algum processo em específico com o seguinte comando:

```
id_do_job = 'COLOCAR_ID_DO_JOP_AQUI'
client.fine_tuning.jobs.retrieve(id_do_job)
```

Você também pode verificar o status por [esta página da OpenAI](#). Clicando em algum dos processo, você pode acompanhar as métricas do treinamento:

Explorando a API da OpenAI

The screenshot shows the OpenAI API dashboard under the 'Fine-tuning' section. On the left, a list of fine-tuning jobs is displayed, with the last job highlighted and a red arrow pointing to it. The right side provides a detailed view of this specific job, including its configuration (Job ID, Base model, Created at, Trained tokens, Epochs, Files), training metrics (Training loss graph), and logs (Messages and Metrics). The logs show the job completed successfully and a new model was created.

Utilizando o novo modelo

Para utilizar o modelo modificado, basta verificar seu nome na página de Fine Tuning:

The screenshot shows the OpenAI API dashboard under the 'Fine-tuning' section. A red arrow points from the 'ft:gpt-3.5-turbo-0125:personal::94CwINIU' entry in the list to the detailed view on the right. The detailed view shows the model configuration, training metrics (including a graph of training loss over epochs), and logs indicating job completion.

E agora utilizar o método `chat.completions`, mas com o argumento `model` definido com o nome do meu novo modelo:

```
mensagens = [
    {'role': 'user', 'content': 'O que é uma equação quadrática?'}
]
```

```
resposta = client.chat.completions.create(  
    messages=mensagens,  
    model='ft:gpt-3.5-turbo-0125:personal::94CwIN1U',  
    max_tokens=1000,  
    temperature=0,  
)  
  
mensagem_resp = resposta.choices[0].message  
print(mensagem_resp.content)  
  
{  
    "resposta": "Uma equação quadrática é uma equação polinomial de segundo grau,"  
    "categoria": "Matemática",  
    "fonte": "AsimoBot"  
}
```

E é possível observar que o modelo respondeu no formato solicitado, em um JSON contendo as chaves resposta, categoria e fonte. Além disso, a resposta possui o tamanho esperado, com um parágrafo contendo até 20 palavras.

Comparando o modelo com Fine-Tuning e o modelo padrão

Se utilizamos a mesma lista de mensagens utilizado com o modelo novo no gpt padrão, a resposta seria a seguinte:

```
mensagens = [  
    {'role': 'user', 'content': 'O que é uma equação quadrática?'}  
]  
  
resposta = client.chat.completions.create(  
    messages=mensagens,  
    model='gpt-3.5-turbo-0125',  
    max_tokens=1000,  
    temperature=0,  
)  
  
mensagem_resp = resposta.choices[0].message  
print(mensagem_resp.content)
```

Uma equação quadrática é uma equação polinomial de segundo grau, ou seja, uma eq

Para obter uma resposta na formatação desejada, teríamos que adicionar o seguinte *prompt*:

```
system_mes = '''  
Responda as perguntas em um parágrafo de até 20 palavras. Categorize as respostas no seguintes  
→ conteúdos: física, matemática, língua portuguesa ou outros.  
'''
```

Explorando a API da OpenAI

```
Retorne a resposta em um formato json, com as keys:  
fonte: valor deve ser sempre AsimoBot  
resposta: a resposta para a pergunta  
categoria: a categoria da pergunta  
'''  
  
mensagens = [  
    {"role": "system", "content": system_mes},  
    {"role": "user", "content": 'O que é uma equação quadrática?'}  
]  
  
resposta = client.chat.completions.create(  
    messages=mensagens,  
    model='gpt-3.5-turbo-0125',  
    max_tokens=1000,  
    temperature=0,  
)  
  
mensagem_resp = resposta.choices[0].message  
print(mensagem_resp.content)  
  
{  
    "fonte": "AsimoBot",  
    "resposta": "Uma equação quadrática é uma equação do segundo grau, ou seja,  
    "categoria": "matemática"  
}
```

Podemos notar que conseguimos economizar vários tokens e mandar mensagens bem menores para atingir o mesmo resultado com o nosso novo modelo!

13. Apresentando a Assistants API para geração de texto avançada

A Assistants API foi projetada para ajudar os desenvolvedores a construir assistentes de IA poderosos capazes de realizar uma variedade de tarefas.

Atenção! A Assistants API está em fase beta e será modificada em breve, à medida que novas funcionalidades surgirem.

- Os assistentes podem chamar os modelos da OpenAI com instruções específicas para ajustar sua personalidade e capacidades.
- Os assistentes podem acessar várias ferramentas em paralelo, incluindo ferramentas hospedadas pela OpenAI, como interpretador de código (Code interpreter) e Knowledge Retrieval, ou ferramentas que você construiu (via chamada de função).
- Os assistentes podem acessar Threads persistentes, que simplificam o desenvolvimento de aplicativos de IA armazenando o histórico de mensagens e truncando-o quando a conversa fica muito longa para o comprimento do contexto do modelo. Você cria um Thread uma vez e simplesmente adiciona mensagens a ele à medida que seus usuários respondem.
- Os assistentes podem acessar arquivos em vários formatos, seja como parte de sua criação ou como parte dos Threads entre os assistentes e os usuários. Ao usar ferramentas, os assistentes também podem criar arquivos (por exemplo, imagens, planilhas, etc) e citar arquivos aos quais fazem referência nas mensagens que criam.

Criando um Assistant

Vamos dar um exemplo rápido de utilização da Assistants. Primeiramente, criamos um novo Assistente com uma instrução específica:

```
from openai import OpenAI
client = OpenAI()

assistant = client.beta.assistants.create(
    name="Math Tutor",
    instructions="You are a personal math tutor. Write and run code to answer math questions.",
    tools=[{"type": "code_interpreter"}],
    model="gpt-4-turbo-preview",
)
```

Nesse caso, ele terá como tarefa responder dúvidas de matemática. Podemos ver que demos acesso à ferramenta de interpretação de código (tools=[{"type": "code_interpreter"}]), ou seja, o assistente será capaz de rodar códigos em Python para resolver as dúvidas.

Uma vez criado, é possível acessar todos os Assistants através da [interface da OpenAI](#).

Explorando a API da OpenAI

The screenshot shows the OpenAI Platform interface at platform.openai.com/assistants. The left sidebar has a navigation menu with items like 'Playground', 'Assistants' (which is highlighted with a red arrow), 'Fine-tuning', 'API keys', 'Storage', 'Usage', 'Settings', 'Documentation', 'Help', 'All products', and 'Personal'. The main content area is titled 'Assistants' and displays a list of existing assistants. Each entry includes the name, ID, and creation timestamp. A message at the bottom says 'Select an assistant to view details'.

Name	ID	Created
Analista de Demonstrações Financeiras	asst_7DP8aa06yKnbMVUjEy5z7B9m	15:00
Analista Financeiro Supermercados Asimov	asst_Grlq7cQdFpIMTSghgIAp	13:59
Analista Financeiro Asimov Supermercados	asst_zVHLJdeVcIgeA2NCG4Hmgt	12:03
Tutor de Matemática da Asimov	asst_joCeCkZlP6VKXNtmgz2wQKCh	11:28
Há 9 dias, 19 de mar.		
Untitled assistant	asst_b70UfUb7RKGZ4A9G1wCLESTG	16:17
Analista de demonstrações financeiras	asst_LBuvWKLchXNoyjmUEjQBSJJC	16:04
Analista de demonstrações financeiras	asst_n6eGPMLmDbYTTES5k8YcmG	15:57
Analista Financeiro Supermercado	asst_cmfpmpPEHfFeJnXZOAdipp	14:53
Tutor de Matemática	asst_BUP2PzTAclctelAk2nF2C	13:56

Criando um Thread

Como mencionado anteriormente, a comunicação com os assistentes é feita através de Threads. As Threads simplificam o desenvolvimento de aplicações de IA ao armazenar o histórico de mensagens e truncá-lo quando a conversa fica muito longa para o comprimento do contexto do modelo. Você cria uma thread uma vez e simplesmente adiciona mensagens a ela conforme o usuário responde. A seguir, vamos criar uma thread:

```
thread = client.beta.threads.create()
```

Adicionando mensagem a Thread

Para adicionar mensagens à thread, basta passar os parâmetros de `thread_id` (que capturamos da thread que acabamos de criar), o `role` e o `content`. Esses dois últimos nós já conhecemos da geração de texto.

```
message = client.beta.threads.messages.create(  
    thread_id=thread.id,  
    role='user',  
    content='Se eu jogar um dado honesto 1000 vezes, qual é a probabilidade de eu obter  
→ exatamente 150 vezes o número 6? Resolva com um código'  
)
```

Solicitando ao Assistant para rodar uma Thread

Por fim, temos que avisar ao Assistant que queremos rodar aquela thread de mensagens que criamos:

```
run = client.beta.threads.runs.create(  
    thread_id=thread.id,  
    assistant_id=assistant.id,  
    instructions='O nome do usuário é Adriano Soares e ele é um usuário Premium.'  
)
```

Além disso, podemos enviar instruções específicas relacionadas àquela rodada de thread. Essa instrução pode melhorar a resposta do Assistente ao aprimorar o contexto para aquele conjunto de mensagens específico. No caso, passamos informações sobre o usuário que fez a solicitação.

Esperando a Thread rodar

Para garantir que a thread já tenha sido executada, antes de prosseguirmos no código, podemos utilizar o seguinte trecho de código:

```
import time  
  
while run.status in ['queued', 'in_progress', 'cancelling']:  
    time.sleep(1)  
    run = client.beta.threads.runs.retrieve(  
        thread_id=thread.id,  
        run_id=run.id  
    )  
    print(run.status) # 'completed'
```

Verificando a resposta

Por fim, basta verificarmos a resposta dada pelo modelo. Para isso, precisamos solicitar à API a lista de mensagens atual da thread:

```
if run.status == 'completed':  
    mensagens = client.beta.threads.messages.list(  
        thread_id=thread.id  
    )  
    print(mensagens)  
else:  
    print('Erro', run.status)
```

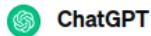
Tendo a mensagem final o formato de texto, podemos verificar-la da seguinte forma:

```
print(mensagens.data[0].content[0].text.value)
```

A probabilidade de obter exatamente 150 vezes o número 6 ao lançar um dado honesto 1000 vezes → é aproximadamente 0.0126, ou seja, cerca de 1.26%.

Analisando os passos de processamento do Assistant

A resposta final foi muito boa e está correta. Inclusive, se fizermos o mesmo questionamento para o ChatGPT normal, ele não conseguirá dar a resposta correta.



ChatGPT

Para calcular a probabilidade de obter exatamente 150 vezes o número 6 ao jogar um dado honesto 1000 vezes, podemos usar a distribuição binomial. A fórmula da distribuição binomial é:

$$P(X = k) = \binom{n}{k} \times p^k \times (1 - p)^{n-k}$$

Onde:

- n é o número total de tentativas (neste caso, 1000 jogadas do dado).
- k é o número de sucessos desejados (neste caso, 150 vezes o número 6).
- p é a probabilidade de sucesso em uma única tentativa (neste caso, a probabilidade de obter o número 6 ao jogar um dado honesto, que é $\frac{1}{6}$).

Podemos usar Python para calcular essa probabilidade. Aqui está o código:

```
python
Copy code

from math import comb

n = 1000 # número total de tentativas
k = 150 # número de sucessos desejados
p = 1/6 # probabilidade de sucesso em uma única tentativa

probabilidade = comb(n, k) * (p**k) * ((1-p)**(n-k))
print(f'A probabilidade de obter exatamente 150 vezes o número 6 em 1000 jogadas é a')
```

Este código calcula a probabilidade usando a função `comb` do módulo `math` para calcular o coeficiente binomial $\binom{n}{k}$. A saída será a probabilidade desejada.



Ele até chega à fórmula correta, mas não consegue executá-la para obter o resultado final. No nosso caso, a fórmula foi executada pelo Assistente em um código Python e, por isso, obtivemos a resposta. Podemos verificar isso ao solicitar todos os passos que o modelo percorreu até chegar à solução.

Primeiro solicitamos todos os passos (*steps*) realizados pelo nosso run:

```
run_steps = client.beta.threads.runs.steps.list(
    thread_id=thread.id,
    run_id=run.id
)
```

Depois analisamos cada passo, para observar o que foi feito:

```
for step in run_steps.data[::-1]:
    print('\n--- Step:', step.step_details.type)
    if step.step_details.type == 'tool_calls':
        for tool_call in step.step_details.tool_calls:
            print('----')
            print(tool_call.code_interpreter.input)
            print('----')
            print('Result')
            print(tool_call.code_interpreter.outputs[0].logs)
    if step.step_details.type == 'message_creation':
        message = client.beta.threads.messages.retrieve(
            thread_id=thread.id,
            message_id=step.step_details.message_creation.message_id
        )
        print(message.content[0].text.value)

--- Step: message_creation
Para calcular a probabilidade de obter exatamente 150 vezes o número 6 ao lançar um dado
→ honesto 1000 vezes, podemos usar a distribuição binomial. A fórmula para a probabilidade
→ em uma distribuição binomial é dada por:
```

$$P(X = k) = \binom{n}{k} \times p^k \times q^{n-k}$$

onde:

- n é o número total de tentativas (lançamentos do dado),
- k é o número de sucessos desejado (número 6),
- p é a probabilidade de sucesso em uma única tentativa,
- q é a probabilidade de fracasso em uma única tentativa ($1 - p$).

Neste caso, $n = 1000$ (número total de lançamentos do dado), $k = 150$ (número de vezes em que queremos obter o número 6), $p = \frac{1}{6}$ (probabilidade de obter o número 6 em um único lançamento) e $q = 1 - p$.

Vamos calcular essa probabilidade com um código Python:

```
--- Step: tool_calls
-----
from math import comb

# Definindo os parâmetros
n = 1000 # número total de lançamentos do dado
k = 150 # número de vezes que queremos obter o número 6
p = 1/6 # probabilidade de obter o número 6 em um único lançamento
q = 1 - p # probabilidade de não obter o número 6 em um único lançamento

# Calculando a probabilidade usando a fórmula da distribuição binomial
probabilidade = comb(n, k) * (p ** k) * (q ** (n - k))
probabilidade
-----
Result
0.01262946340594314
```

```
==== Step: message_creation
A probabilidade de obter exatamente 150 vezes o número 6 ao lançar um dado honesto 1000 vezes
→ é aproximadamente 0.0126, ou seja, cerca de 1.26%.
```

Podemos ver que ele necessitou de três passos:

- Criação de mensagem (message_creation): ele verificou a fórmula necessária para conseguir chegar a solução.
- Chamamento de função (tool_calls): ele executou o código em Python relativo a fórmula anterior obtida
- Criação de mensagem (message_creation): gerou a resposta final para o usuário.

Espero que este exemplo elucidde um pouco das capacidades dos Assistants e como as suas ferramentas externas podem aumentar seu poder.

14. Analisando dados com Assistants Code Interpreter

As habilidades de executar um código adicionam enormes capacidades ao Assistente. Além de criar códigos simples, ele pode interagir com arquivos passados a ele e extrair insights relevantes e rápidos, com poucas linhas de código. E é isso que vamos explorar aqui brevemente: análise de dados com Assistants.

Primeiro, vamos olhar os dados que vamos utilizar. Ele são referentes a vendas de um supermercado, e foram extraídos do [Kaggle](#)

```
import pandas as pd
```

```
dataset = pd.read_csv('arquivos/supermarket_sales.csv')
dataset.head()
```

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax %	Total	Date	Time	Payment	cogs	gross margin percentage	gross income	Rating
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548.9715	1/5/2019	13:08	Ewallet	522.83	4.761905	26.1415	9.1
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.2200	3/8/2019	10:29	Cash	76.40	4.761905	3.8200	9.6
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.5255	3/3/2019	13:23	Credit card	324.31	4.761905	16.2155	7.4
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.0480	1/27/2019	20:33	Ewallet	465.76	4.761905	23.2880	8.4
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.3785	2/8/2019	10:37	Ewallet	604.17	4.761905	30.2085	5.3

Enviando arquivos para o Assistant

Agora vamos criar um novo assistant que possui acesso aos meus dados:

```
import openai
from dotenv import load_dotenv, find_dotenv

_ = load_dotenv(find_dotenv())

client = openai.Client()

file = client.files.create(
    file=open('arquivos/supermarket_sales.csv', 'rb'),
    purpose='assistants'
)

assistant = client.beta.assistants.create(
    name="Analista Financeiro Supermercados Asimov",
    instructions="Você é um analista financeiro de um supermercado. \
    Você utiliza os dados .csv relativo às vendas \
    do supermercado para realizar as suas análises",
    tools=[{'type': 'code_interpreter'}],
    file_ids=[file.id],
    model='gpt-4-turbo-preview'
```

Você pode notar que existe um parâmetro file_ids na criação de assistentes que recebe uma lista de IDs como argumento (file_ids=[file.id],). Podemos adicionar até 10 documentos diferentes a um assistente.

Agora vamos fazer a seguinte pergunta ao Assistente:

Qual é o rating médio das vendas do nosso supermercado?

Para isso, precisamos criar uma Thread, adicionar uma mensagem a ela, solicitar que o assistente execute a thread e esperar que o processo finalize.

```
# Cria thread
thread = client.beta.threads.create()

# Adiciona mensagem
pergunta = 'Qual é o rating médio das vendas do nosso supermercado?'
messages = client.beta.threads.messages.create(
    thread_id=thread.id,
    role='user',
    content=pergunta
)

# Roda thread
run = client.beta.threads.runs.create(
    thread_id=thread.id,
    assistant_id=assistant.id,
    instructions='O nome do usuário é Adriano.'
)

# Espera ela rodar
import time

while run.status in ['queued', 'in_progress', 'cancelling']:
    time.sleep(1)
    run = client.beta.threads.runs.retrieve(
        thread_id=thread.id,
        run_id=run.id
    )

print(run.status)
```

Agora podemos verificar a resposta:

```
if run.status == 'completed':
    messages = client.beta.threads.messages.list(
        thread_id=thread.id
    )
    print(messages)
else:
    print('Erro', run.status)
```

E analisando a resposta:

```
print(messages.data[0].content[0].text.value)

0 rating médio das vendas do seu supermercado é aproximadamente 6.97. Se precisar de mais
→ informações ou análises adicionais, por favor, me avise!
```

Podemos verificar que ele chegou a resposta correta!

```
dataset['Rating'].mean()
```

6.9727

Verificando os passos

Para verificar os passos necessários para chegar à resposta, basta executar o trecho de código do capítulo anterior.

```
run_steps = client.beta.threads.runs.steps.list(
    thread_id=thread.id,
    run_id=run.id
)

for step in run_steps.data[::-1]:
    print('===== Step >', step.step_details.type)
    if step.step_details.type == 'tool_calls':
        for tool_call in step.step_details.tool_calls:
            print('---')
            print(tool_call.code_interpreter.input)
            print('---')
            if tool_call.code_interpreter.outputs[0].type == 'logs':
                print('Result')
                print(tool_call.code_interpreter.outputs[0].logs)
    if step.step_details.type == 'message_creation':
        message = client.beta.threads.messages.retrieve(
            thread_id=thread.id,
            message_id=step.step_details.message_creation.message_id
        )
        if message.content[0].type == 'text':
            print(message.content[0].text.value)

===== Step > message_creation
Para calcular o rating médio das vendas do seu supermercado, primeiro precisarei analisar o
→ arquivo que você enviou. Vou começar por abrir e examinar o conteúdo do arquivo para
→ identificar a estrutura dos dados e a coluna que contém os ratings das vendas.
===== Step > tool_calls

import pandas as pd

# Carregar o arquivo
file_path = '/mnt/data/file-cEUCgJzyV026Y5fZb4erTPqt'
data = pd.read_csv(file_path)

# Exibir as primeiras linhas para entender a estrutura dos dados
```

Explorando a API da OpenAI

```
data.head()

Result
      Invoice ID Branch      City Customer type  Gender \
0  750-67-8428        A    Yangon       Member Female
1  226-31-3081        C  Naypyitaw     Normal Female
2  631-41-3108        A    Yangon     Normal Male
3  123-19-1176        A    Yangon       Member Male
4  373-73-7910        A    Yangon     Normal Male

      Product line  Unit price  Quantity    Tax 5%      Total      Date \
0  Health and beauty    74.69       7  26.1415  548.9715  1/5/2019
1  Electronic accessories   15.28       5   3.8200  80.2200  3/8/2019
2  Home and lifestyle     46.33       7  16.2155  340.5255  3/3/2019
3  Health and beauty     58.22       8  23.2880  489.0480  1/27/2019
4  Sports and travel     86.31       7  30.2085  634.3785  2/8/2019

      Time      Payment      cogs gross margin percentage  gross income  Rating
0  13:08    Ewallet  522.83           4.761905  26.1415    9.1
1  10:29      Cash   76.40           4.761905   3.8200    9.6
2  13:23  Credit card  324.31           4.761905  16.2155    7.4
3  20:33    Ewallet  465.76           4.761905  23.2880    8.4
4  10:37    Ewallet  604.17           4.761905  30.2085    5.3

===== Step > message_creation
0 arquivo contém várias colunas, incluindo 'Rating', que parece ser a coluna relevante para
→ calcular o rating médio das vendas do supermercado. Vou proceder com o cálculo do rating
→ médio agora.
===== Step > tool_calls

# Calcular o rating médio
average_rating = data['Rating'].mean()
average_rating

Result
6.9727
===== Step > message_creation
0 rating médio das vendas do seu supermercado é aproximadamente 6.97. Se precisar de mais
→ informações ou análises adicionais, por favor, me avise!
```

Podemos ver que ele necessitou de cinco passos:

- Criação de mensagem (message_creation): ele percebeu que era necessário abrir o arquivo e verificar as informações contidas nele.
- Chamada de função (tool_calls): ele executou o código em Python para abrir o arquivo e verificar os primeiros dados.
- Criação de mensagem (message_creation): ele percebeu que havia uma coluna “Rating” e a partir dela seria possível calcular o que o usuário solicitou.
- Chamada de função (tool_calls): ele executou o código em Python para calcular o Rating a partir do arquivo informado.
- Criação de mensagem (message_creation): gerou a resposta final para o usuário.

Gerando gráficos com Assistants

Outra feature bem legal é a possibilidade de gerar gráficos ao utilizarmos o code_interpreter. Vamos ver como isso funciona.

Primeiro, vamos solicitar o seguinte ao modelo:

Gere um gráfico pizza com o percentual de vendas por meio de pagamento

```
# Adiciona mensagem a thread
pergunta = 'Gere um gráfico pizza com o percentual de vendas por meio de pagamento'

messages = client.beta.threads.messages.create(
    thread_id=thread.id,
    role='user',
    content=pergunta
)

# Solicita ao assistente que rode a thread
run = client.beta.threads.runs.create(
    thread_id=thread.id,
    assistant_id=assistant.id,
    instructions='O nome do usuário é Adriano.'
)

# Aguarda a thread rodar
import time

while run.status in ['queued', 'in_progress', 'cancelling']:
    time.sleep(1)
    run = client.beta.threads.runs.retrieve(
        thread_id=thread.id,
        run_id=run.id
    )

print(run.status)
```

Verificamos as mensagens e notamos que a última não consiste mais em um texto, e sim em um arquivo de imagem (image_file).

```
if run.status == 'completed':
    messages = client.beta.threads.messages.list(
        thread_id=thread.id
    )
    print(messages)
else:
    print('Erro', run.status)

print(messages.data[0].content[0])
```

Podemos agora verificar os passos do modelo:

```
for step in run_steps.data[::-1]:
    print('===== Step >', step.step_details.type)
    if step.step_details.type == 'tool_calls':
        for tool_call in step.step_details.tool_calls:
            print('---')
            print(tool_call.code_interpreter.input)
            print('---')
            if tool_call.code_interpreter.outputs[0].type == 'logs':
                print('Result')
                print(tool_call.code_interpreter.outputs[0].logs)
    if step.step_details.type == 'message_creation':
        message = client.beta.threads.messages.retrieve(
            thread_id=thread.id,
            message_id=step.step_details.message_creation.message_id
        )
        if message.content[0].type == 'text':
            print(message.content[0].text.value)

        if message.content[0].type == 'image_file':
            file_id = message.content[0].image_file.file_id
            image_data = client.files.content(file_id)

            with open(f'arquivos/{file_id}.png', 'wb') as file:
                file.write(image_data.read())

            import matplotlib.pyplot as plt
            import matplotlib.image as mpimg

            img = mpimg.imread(f'arquivos/{file_id}.png')
            fig, ax = plt.subplots()
            ax.set_axis_off()
            ax.imshow(img)
            plt.show()

===== Step > tool_calls

import matplotlib.pyplot as plt

# Agregar a quantidade de vendas por meio de pagamento
pagamento_counts = data['Payment'].value_counts()

# Gerar o gráfico de pizza
plt.figure(figsize=(10, 7))
plt.pie(pagamento_counts, labels=pagamento_counts.index, autopct='%1.1f%%', startangle=140)
plt.title('Percentual de Vendas Por Meio de Pagamento')
plt.show()

===== Step > message_creation
```



E ele gerou corretamente o gráfico solicitado. Incrível!

15. Analisando arquivos pdf com Assistants Retrieval

A última ferramenta dos Assistants que vamos explorar é a obtenção de informações (ou *Knowledge Retrieval*). Através dela, o Assistente aumenta suas capacidades com conhecimento de fora de seu modelo, como informações de produtos proprietários ou documentos fornecidos pelos seus usuários. Assim que um arquivo é carregado e passado para o Assistente, a OpenAI automaticamente segmentará seus documentos, indexará e armazenará os embeddings, e implementará uma busca vetorial para recuperar conteúdo relevante para responder às consultas dos usuários.

No exemplo a seguir, passaremos ao Assistente um arquivo PDF com as demonstrações de resultado para a empresa Ambev e faremos perguntas, onde ele terá que processar o arquivo para nos retornar uma resposta.

Começamos criando o Assistant:

```
import openai
from dotenv import load_dotenv, find_dotenv

_ = load_dotenv(find_dotenv())

client = openai.Client()

# Enviando o arquivo para o servidor da OpenAI
file = client.files.create(
    file=open('arquivos/divulgacao_resultado_ambev_4T23.pdf', 'rb'),
    purpose='assistants'
)

# Criando o assistant
assistant = client.beta.assistants.create(
    name="Analista de Demonstrações Financeiras",
    instructions="Você é um analista de demonstrações \
        financeiras da Ambev. Você tem acesso a demonstração \
        de resultado do 4º trimestre de 2023. Baseado apenas \
        no documento que você tem acesso, responda \
        as perguntas do usuário.",
    tools=[{'type': 'retrieval'}],
    file_ids=[file.id],
    model='gpt-4-turbo-preview'
)
```

Nas ferramentas agora, adicionamos o retrieval (tools=[{"type": "retrieval"}]), avisando a API que gostaríamos de utilizá-la.

E agora enviamos uma mensgem:

```
# Criamos uma thread
thread = client.beta.threads.create()

# Adicionamos uma mensagem a thread
```

Explorando a API da OpenAI

```
pergunta = 'Qual o volume de cerveja vendido no Brasil segundo o documento?'

messages = client.beta.threads.messages.create(
    thread_id=thread.id,
    role='user',
    content=pergunta
)

# Solicitamos ao assistente que rode a thread
run = client.beta.threads.runs.create(
    thread_id=thread.id,
    assistant_id=assistant.id,
    instructions='O nome do usuário é Adriano.'
)

# Aguradamos a finalização do processo
import time

while run.status in ['queued', 'in_progress', 'cancelling']:
    time.sleep(1)
    run = client.beta.threads.runs.retrieve(
        thread_id=thread.id,
        run_id=run.id
    )

print(run.status)
```

Podemos ver que a resposta foi:

```
print(messages.data[0].content[0].text.value)

Segundo o documento, o volume de cerveja vendido no Brasil pela Ambev no ano de 2023 foi de
→ 183.659 mil hectolitros, o que representa uma redução de 11% em relação ao ano anterior,
→ 2022, quando o volume foi de 185.7497 mil hectolitros [7+source].
```

O símbolo [7+source] mostra que o modelo utilizou informações adquiridas através de retrieval para gerar aquela parte da resposta. Podemos analisar os passos, da mesma forma que fazímos antes:

```
run_steps = client.beta.threads.runs.steps.list(
    thread_id=thread.id,
    run_id=run.id
)

for step in run_steps.data[::-1]:
    print('===== Step >', step.step_details.type)
    if step.step_details.type == 'tool_calls':
        for tool_call in step.step_details.tool_calls:
            if tool_call.type == 'retrieval':
                print(tool_call)
    if step.step_details.type == 'message_creation':
        message = client.beta.threads.messages.retrieve(
            thread_id=thread.id,
            message_id=step.step_details.message_creation.message_id
        )
```

Explorando a API da OpenAI

```
if message.content[0].type == 'text':
    message = client.beta.threads.messages.retrieve(
        thread_id=thread.id,
        message_id=step.step_details.message_creation.message_id
    )
    print(message.content[0].text.value)

===== Step > tool_calls
RetrievalToolCall(id='call_nMmvggvJM4iVGSEEx8UFN3l05', retrieval={}, type='retrieval')
===== Step > message_creation
Segundo o documento, o volume de cerveja vendido no Brasil pela Ambev no ano de 2023 foi de
↪ 183.659 mil hectolitros, o que representa uma redução de 11% em relação ao ano anterior,
↪ 2022, quando o volume foi de 185.7497 mil hectolitros [7+source].
```

Podemos ver que dois passos forma necessários, primeiro um tool_calls de retrieval e depois a formulação da resposta final.

16. Criando e editando imagens com Dall-e

Já falamos bastante sobre geração de texto e as ferramentas mais avançadas que a OpenAI disponibiliza hoje para este propósito. Agora vamos explorar outros campos de utilização da API também de extrema relevância, começando pelos modelos de geração de imagem.

Vamos tratar das 3 funcionalidades de imagem disponíveis hoje:

1. Criar imagens do zero com base em um prompt de texto (DALL-E 3 e DALL-E 2)
2. Criar versões editadas de imagens tendo o modelo substituir algumas áreas de uma imagem pré-existente, com base em um novo prompt de texto (apenas DALL-E 2)
3. Criar variações de uma imagem existente (apenas DALL-E 2)

Vamos fazer nossos imports como de costume:

```
import requests
from PIL import Image

import openai
from dotenv import load_dotenv, find_dotenv

_ = load_dotenv(find_dotenv())
client = openai.Client()
```

Vocês podem notar que fizemos mais dois imports que serão necessários para conseguirmos visualizar a imagem gerada. Requests já faz parte da biblioteca padrão, enquanto o PIL (que vem de Pillow) não, e precisa ser instalado:

```
pip install Pillow
```

Criando uma imagem

Feito isso, podemos gerar as imagens:

```
nome = 'bosque'
modelo = 'dall-e-3'
prompt = 'Crie uma imagem de um campo de pastagem, \
          amplo com uma leve elevação ao fundo.'
qualidade = 'hd'
style = 'natural'

resposta = client.images.generate(
    model=modelo,
    prompt=prompt,
    size='1024x1024',
    quality=qualidade,
```

```
    style=style,  
    n=1  
)
```

Temos os seguintes argumentos para explorar no método images.generate:

- **model**: o modelo utilizado:
 - “dall-e-3”
 - “dall-e-2”
- **prompt**: o comando para geração de imagem
- **size**: o tamanho da imagem final:
 - “256x256”
 - “512x512”
 - “1024x1024”
 - “1792x1024”
 - “1024x1792”
- **quality**: a qualidade da imagem:
 - “standard”
 - “hd”
- **style**: estilo da imagem:
 - “vivid” > imagens hiper-realistas e dramáticas
 - “natural” > imagens mais naturais
- **n**: a quantidade de imagens que serão geradas simultaneamente

Salvando a imagem gerada

Agora vamos salvar a imagem:

```
nome_arquivo = f'{nome}_{modelo}_{qualidade}_{style}.jpg'  
  
image_url = resposta.data[0].url  
img_data = requests.get(image_url).content  
with open(nome_arquivo, 'wb') as f:  
    f.write(img_data)
```

Visualizando a imagem

E utilizamos PIL para visualizar a imagem:

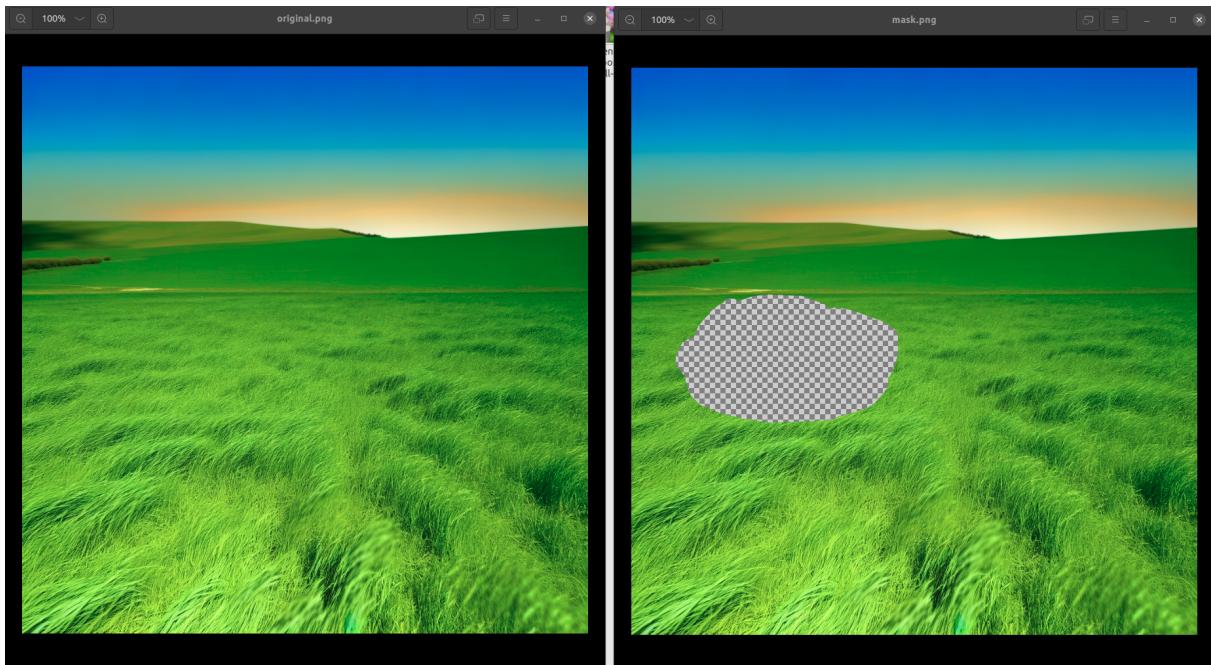
```
image = Image.open(nome_arquivo)  
image.show()
```



Editando uma imagem

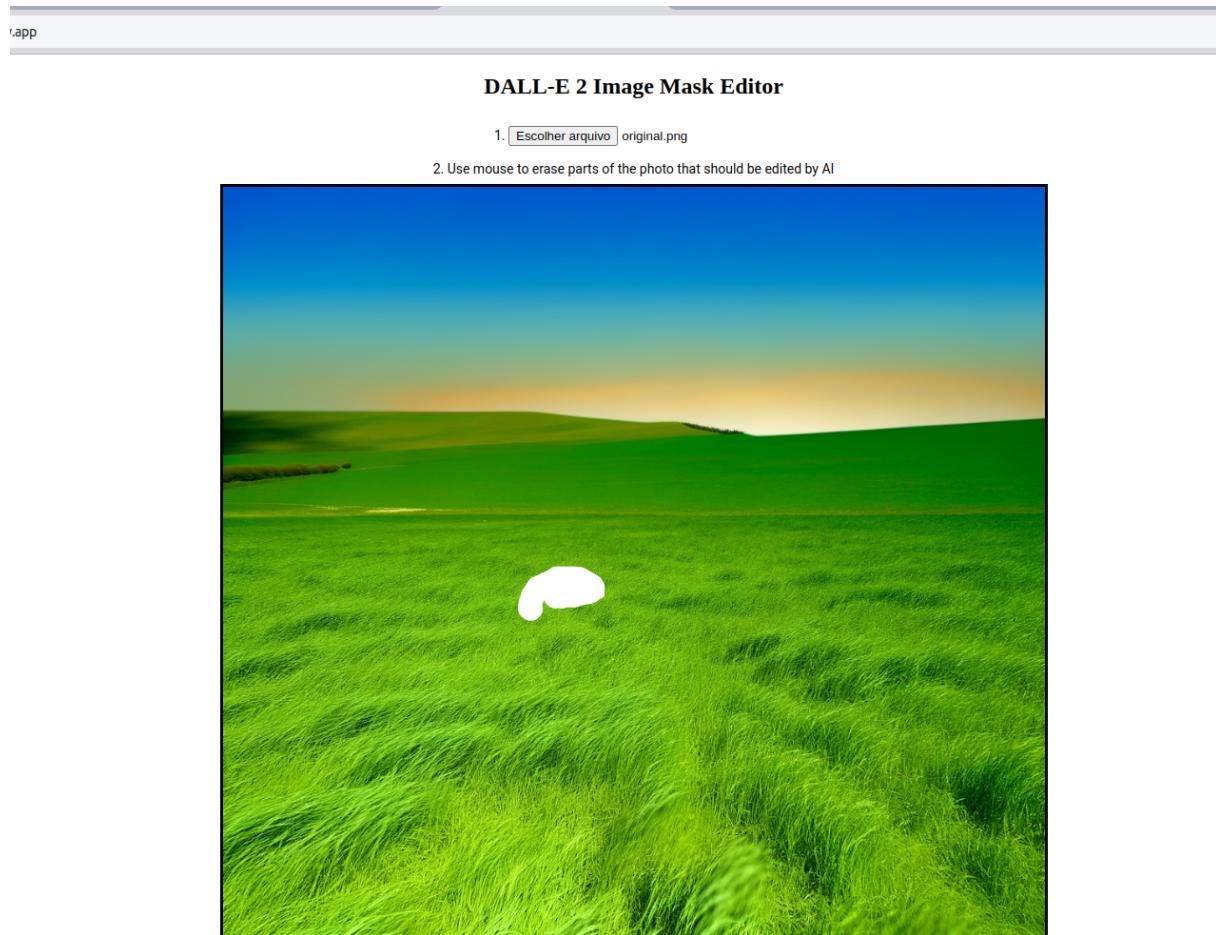
Para editar uma imagem, é necessário primeiro criarmos uma máscara. Ela consiste na mesma imagem no formato PNG, apenas com a parte que será editada em branco, como podemos observar na imagem:

Explorando a API da OpenAI

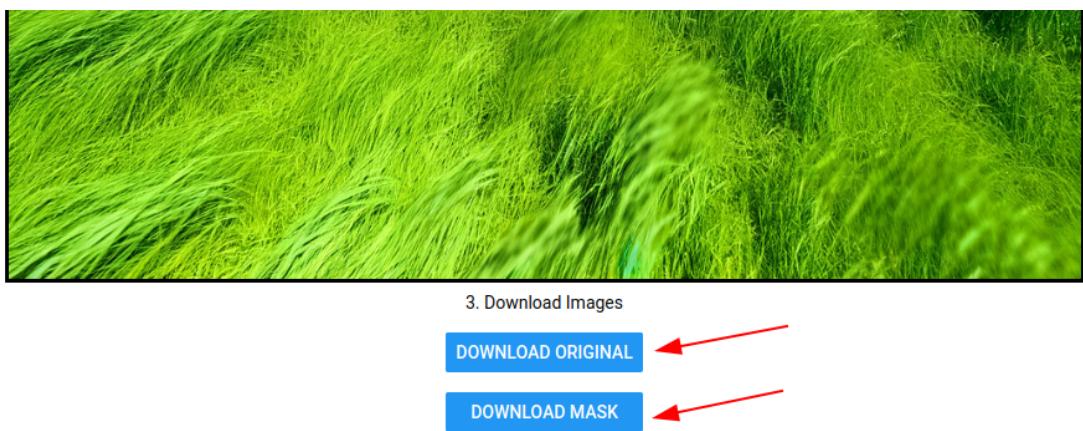


Para gerar a mask, podemos utilizar [este site](#).

Explorando a API da OpenAI



O processo é simples, apenas adicione uma nova imagem. Com o mouse clicado, apague uma parte da imagem e depois faça o download da mask e da original novamente.



Feito isso, podemos voltar para o código:

```
resposta = client.images.edit(  
    model='dall-e-2',
```

```
image=open('arquivos/imagens/original.png', 'rb'),
mask=open('arquivos/imagens/mask.png', 'rb'),
prompt='Adicione uma vaca e um terneirinho na imagem fornecida',
n=1,
size='1024x1024'
)
```

Salvando a imagem gerada

Agora vamos salvar a imagem:

```
nome_arquivo = 'editada.jpg'

image_url = resposta.data[0].url
img_data = requests.get(image_url).content
with open(nome_arquivo, 'wb') as f:
    f.write(img_data)
```

Visualizando a imagem

E utilizamos PIL para visualizar a imagem:

```
image = Image.open(nome_arquivo)
image.show()
```



Criando variações

Por último, podemos criar variações de imagens com o seguinte código:

```
resposta = client.images.create_variation(  
    image=open('arquivos/imagens/bosque_dall-e-3_hd_natural.jpg', 'rb'),  
    n=1,  
    size='1024x1024'  
)
```

Salvando a imagem gerada

Agora vamos salvar a imagem:

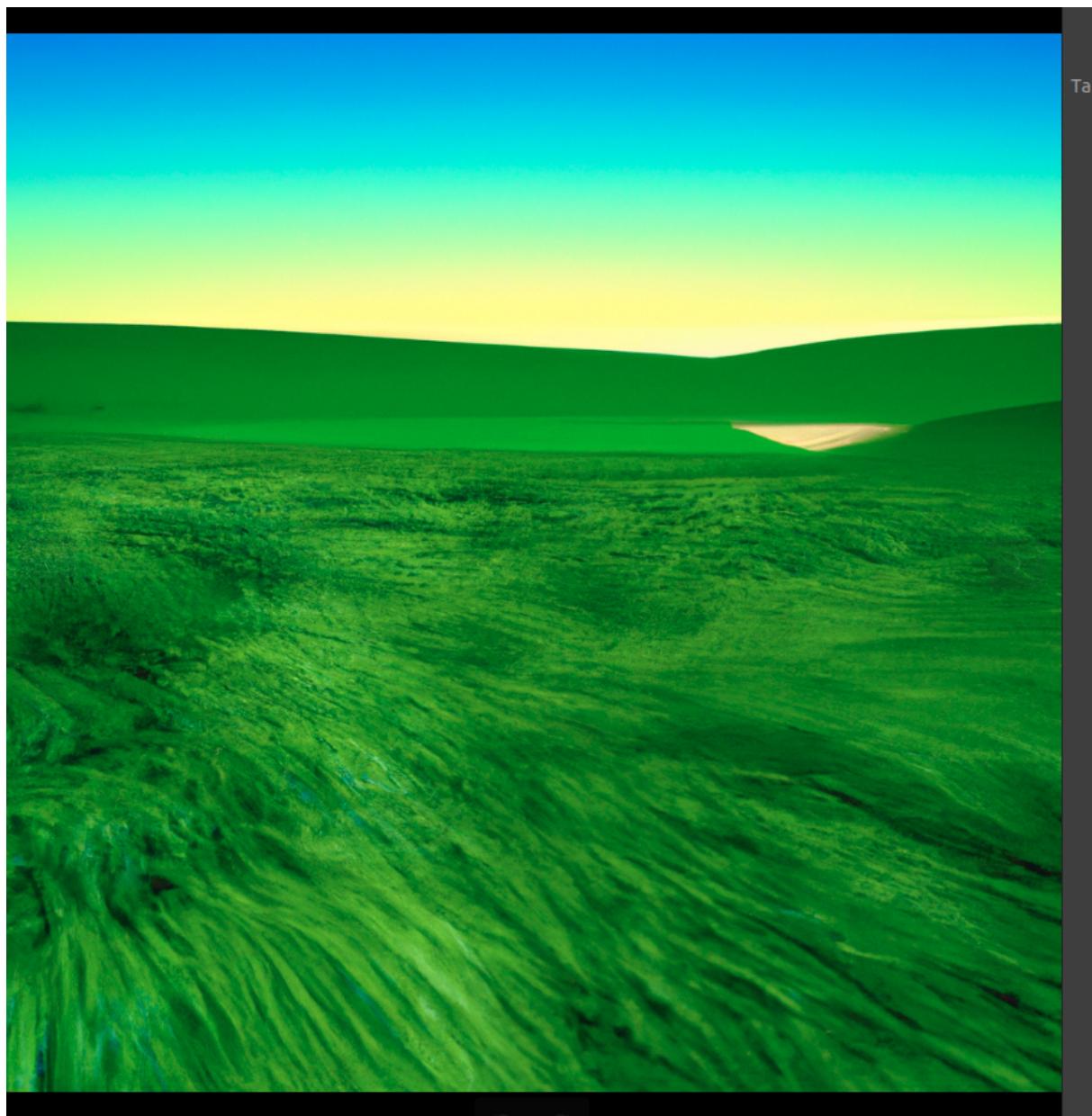
```
nome_arquivo = 'variacao.jpg'

image_url = resposta.data[0].url
img_data = requests.get(image_url).content
with open(nome_arquivo, 'wb') as f:
    f.write(img_data)
```

Visualizando a imagem

E utilizamos PIL para visualizar a imagem:

```
image = Image.open(nome_arquivo)
image.show()
```



17. Visão computacional com GPT-Vision

Mais uma ferramenta muito poderosa desenvolvida pela OpenAI é o GPT-Vision. Ele é um modelo híbrido que, além de conseguir gerar texto, pode interpretar imagens. Ou seja, o modelo recebe imagens e responde perguntas sobre elas. Incrível!

Interpretando uma imagem da internet

Podemos interpretar qualquer imagem hospedada em um site com o seguinte comando:

```
# imports necessários
import openai
from dotenv import load_dotenv, find_dotenv

_ = load_dotenv(find_dotenv())

client = openai.Client()

#interpretando a imagem
comando = 'Descreva a imagem fornecida'
url = 'https://upload.wikimedia.org/wikipedia/commons/thumb/d/dd/Gfp-wisconsin-madison-the-\
→ nature-boardwalk.jpg/2560px-Gfp-wisconsin-madison-the-nature-boardwalk.jpg'

resposta = client.chat.completions.create(
    model='gpt-4-vision-preview',
    messages=[{
        'role': 'user',
        'content': [
            {'type': 'text', 'text': comando},
            {'type': 'image_url', 'image_url': {'url': url}}
        ]
    }]
)
```

Esta é a imagem que estamos tentando descrever:



E a resposta do modelo é a seguinte:

```
print(resposta.choices[0].message.content)
```

A imagem mostra uma paisagem natural serena composta por um céu azul claro com algumas nuvens esparsas. No primeiro plano, há uma passarela de madeira que se estende através de um campo de grama alta e verde, sugerindo que talvez seja uma área úmida ou um parque natural protegido. A passarela parece convidativa, guiando o olhar do espectador através da cena. A grama verde vibrante de ambos os lados da passarela contrasta com o azul do céu. Ao fundo, podem ser observadas árvores e arbustos baixos, indicando a presença de um ecossistema diversificado. A composição da imagem, a luz natural e a paleta de cores vivas criam uma atmosfera de tranquilidade e beleza natural.

Incrível a qualidade da resposta. Poucas pessoas conseguiriam fazer melhor!

Interpretando uma imagem do seu computador

Para enviar uma imagem do seu computador, é necessário antes realizar um encoding para base64:

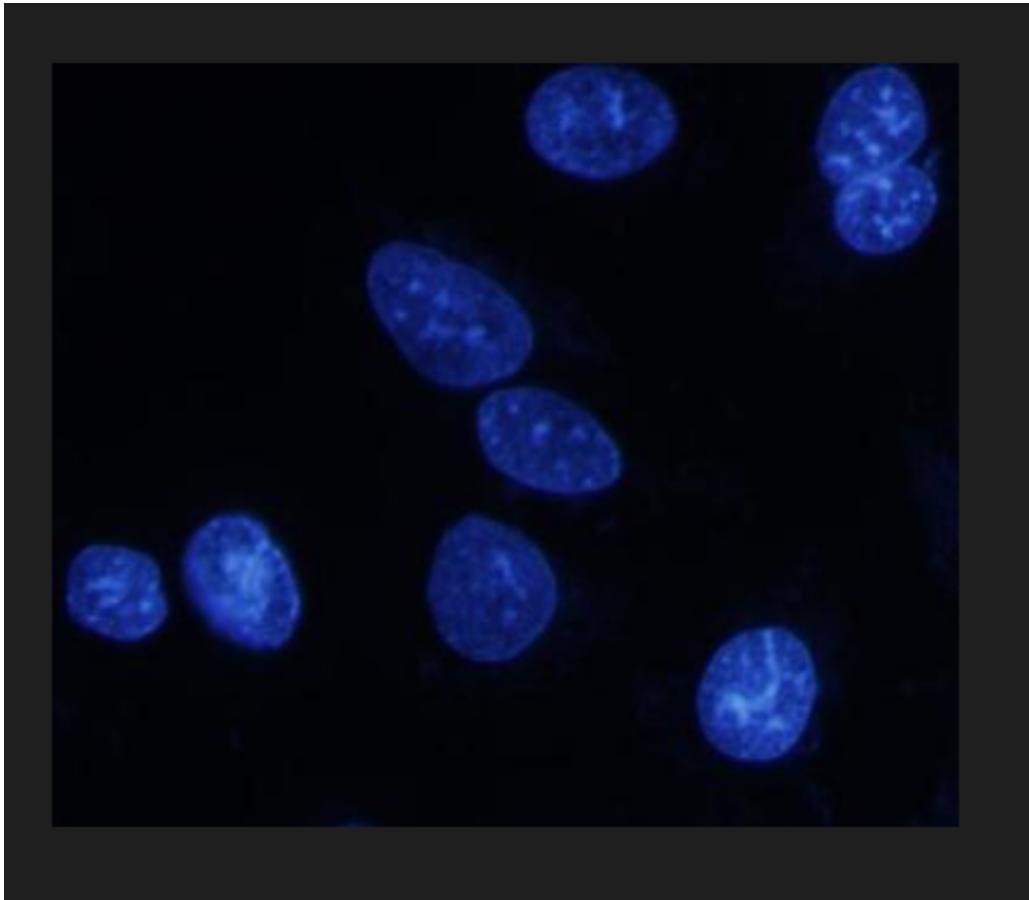
```
import base64

def encode_image(caminho_imagem):
    with open(caminho_imagem, 'rb') as img:
        return base64.b64encode(img.read()).decode('utf-8')

caminho = 'celulas.jpg'
base_64_img = encode_image(caminho)
```

Explorando a API da OpenAI

Esta é a imagem que vamos enviar:



Agora rodamos o modelo perguntando quantas células ele vê na imagem:

```
comando = 'Quantas células aparecem na imagem?'
url = f'data:image/jpg;base64,{base_64_img}'

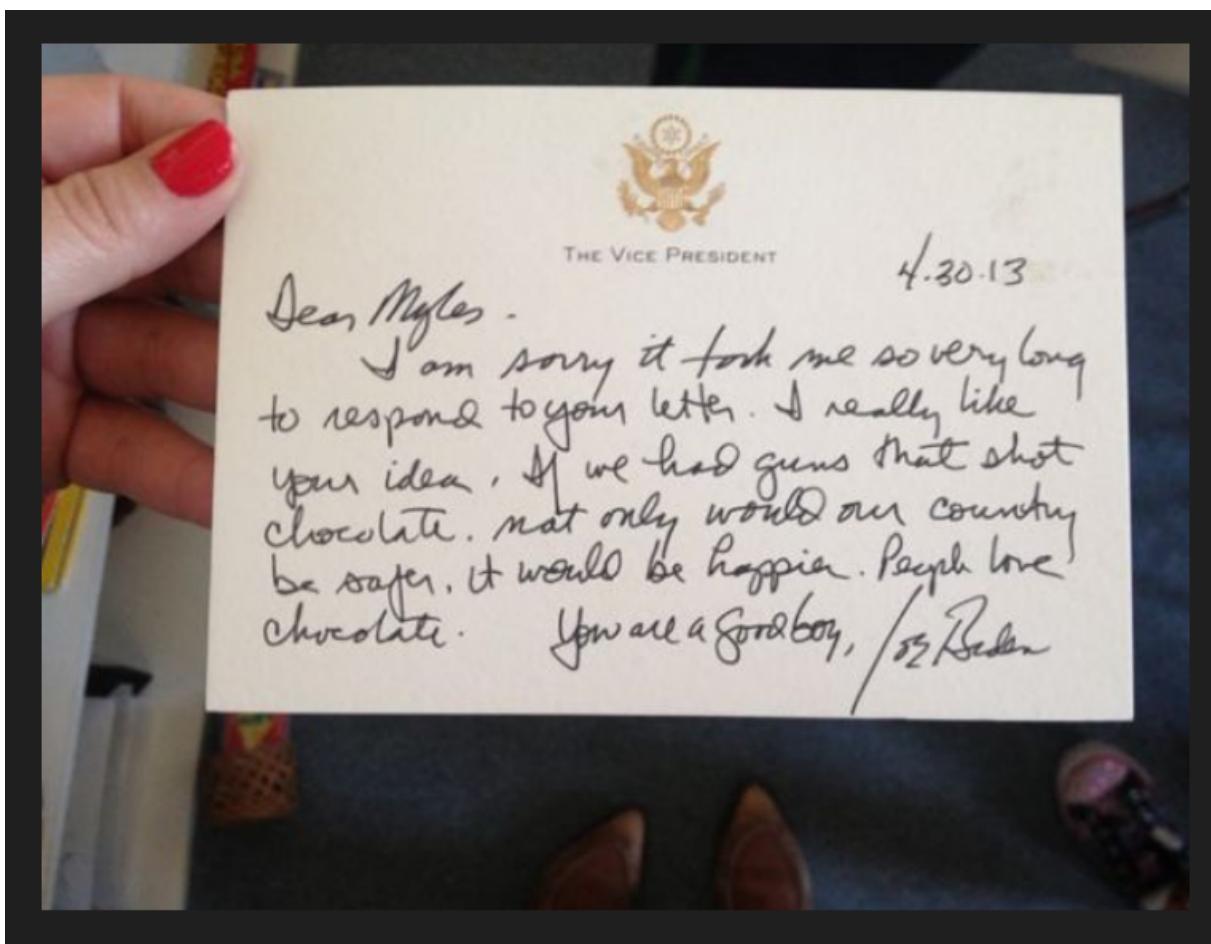
resposta = client.chat.completions.create(
    model='gpt-4-vision-preview',
    messages=[{
        'role': 'user',
        'content': [
            {'type': 'text', 'text': comando},
            {'type': 'image_url', 'image_url':
                {'url': url}}
        ]
    }],
    max_tokens=1000,
)
print(resposta.choices[0].message.content)
```

Essa imagem mostra células que foram marcadas com um corante que destaca seus núcleos. Pela → imagem, parece haver um total de nove núcleos, o que sugere nove células. No entanto, é → importante notar que devido à resolução e ao plano de foco da imagem, algumas células → podem não estar inteiramente visíveis ou podem estar sobrepostas, o que dificulta uma → contagem precisa apenas com base nesta imagem estática.

Muito impressionante. Ele conseguiu identificar prontamente as células.

Interpretando palavras escritas

Agora vamos testar a habilidade do modelo de identificar a palavra escrita. Vamos ver se ele consegue decifrar este texto:



```
import base64

def encode_image(caminho_imagem):
    with open(caminho_imagem, 'rb') as img:
        return base64.b64encode(img.read()).decode('utf-8')

caminho = 'escrito_mao_dificil.jpg'
```

Explorando a API da OpenAI

```
base_64_img = encode_image(caminho)

texto = "O que está escrito na imagem?"

resposta = client.chat.completions.create(
    model='gpt-4-vision-preview',
    messages=[{
        'role': 'user',
        'content': [
            {'type': 'text', 'text': texto},
            {'type': 'image_url', 'image_url':
                {'url': f'data:image/jpg;base64,{base_64_img}'}}
        ]
    }],
    max_tokens=1000,
)

print(resposta.choices[0].message.content)
```

Na imagem, vê-se um cartão com o seguinte texto manuscrito:

The Vice President
4/30/13

Dear Myles,

I am sorry it took me so very long to respond to your letter. I really like your idea. If we
had guns that shot chocolate, **not** only would our country be safer, it would be happier.
People love chocolate.

You are a good boy,

Joe Biden

Esta imagem mostra uma carta escrita pelo então vice-presidente Joe Biden para uma criança
chamada Myles, comentando sobre uma ideia relacionada a armas que disparariam chocolate.

Incrível, ele acertou perfeitamente!

18. Criação de áudios a partir de textos

Já conhecemos as habilidades da API para geração e interpretação de textos e imagens. Agora vamos para sua última grande aplicação: geração e transcrição de áudios. Comecemos com a geração:

```
# imports
import openai
from dotenv import load_dotenv, find_dotenv

_ = load_dotenv(find_dotenv())

client = openai.Client()

# gera áudio
arquivo = 'fala.mp3'
texto = '''
Python é uma linguagem de programação de alto nível, interpretada de script, imperativa,
→ orientada a objetos,
funcional, de tipagem dinâmica e forte. Foi lançada por Guido van Rossum em 1991. Atualmente,
→ possui um modelo
de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos
→ Python Software Foundation.
Apesar de várias partes da linguagem possuírem padrões e especificações formais, a linguagem,
→ como um todo, não é
formalmente especificada. O padrão na prática é a implementação CPython.

A linguagem foi projetada com a filosofia de enfatizar a importância do esforço do programador
→ sobre o esforço
computacional. Prioriza a legibilidade do código sobre a velocidade ou expressividade. Combina
→ uma sintaxe concisa
e clara com os recursos poderosos de sua biblioteca padrão e por módulos e frameworks
→ desenvolvidos por terceiros.
'''

resposta = client.audio.speech.create(
    model='tts-1',
    voice='onyx',
    input=texto
)
resposta.write_to_file(arquivo)
```

Temos os seguintes argumentos para explorar no método `audio.speech.create`:

- `model`: o modelo utilizado:
 - “`tts-1`”
 - “`tts-1-hd`”
- `voice`: o estilo da voz que será usado
 - “`alloy`”

- “echo”
 - “fable”
 - “onyx”
 - “nova”
 - “shimmer”
- `input`: o texto que será transformado em fala

E é simples assim! Os resultados do modelo são de ótima qualidade. Pelos meus testes, a voz que se encaixa melhor com o português do brasil é a “onyx”.

19. Transcrição de áudios

Por fim, chegamos à última função disponível na API da OpenAI: a transcrição de áudios. É possível transcrever áudio de qualquer língua que você quiser, e o modelo reconhecerá o idioma automaticamente.

No exemplo, passamos um áudio de uma das nossas aulas e vamos ver como o modelo se comporta:

```
import openai
from dotenv import load_dotenv, find_dotenv

_ = load_dotenv(find_dotenv())

client = openai.Client()

audio = open('audio_asimov.mp3', 'rb')
transcricao = client.audio.transcriptions.create(
    model='whisper-1',
    file=audio
)
print(transcricao.text)
```

Seja muito bem-vindo ou bem-vinda ao nosso curso completo de Python aqui da Zemove Academy. Eu → e minha equipe ficamos muito felizes que vocês tenham escolhido iniciar no mundo da → programação, especificamente com a linguagem Python, aqui com a gente. Pode ter certeza → que a gente colocou muito carinho e muita dedicação para construir esse material. Além dos → conhecimentos técnicos que a gente vai apresentar sobre a linguagem e programação em si, → eu também coloquei grande parte da minha experiência e minha vivência para compartilhar → com vocês ao longo desse treinamento. Para quem não me conhece ainda, meu nome é Rodrigo → Soares Padeval e eu não sou programador de origem. Na verdade, eu me formei como → engenheiro e eu utilizei a programação dentro da minha carreira no mercado financeiro como → analista de dados. E essa é a grande mágica da programação. Você não precisam utilizá-la → única e exclusivamente para desenvolver software. Na verdade, ela pode ser usada para o → que vocês quiserem no dia a dia de vocês, muitas vezes como uma habilidade secundária. Ela → é muito mais ampla e muito mais poderosa do que isso. Programação é, sem sombra de → dúvidas, a habilidade do futuro. Tem gente que já compara e diz que a programação é o novo → inglês. Ou seja, até anos atrás, era necessário que a gente tivesse inglês para poder → entrar no mercado de trabalho. Hoje, com certeza, além do inglês, a gente tem que saber → programar.

O resultado foi ótimo. Entretanto, ele cometeu alguns erros ao se tratar de nomes próprio. Por exemplo, Asimov Academy virou Zemove Academy e Rodrigo Soares Tadewald virou Rodrigo Soares Padeval. Fez sentido, mas pode melhorar.

Para corrigir, podemos utilizar o parâmetro prompt, da seguinte forma:

```
audio = open('audio_asimov.mp3', 'rb')
transcricao = client.audio.transcriptions.create(
    model='whisper-1',
    file=audio,
    prompt='Essa é a transcrição de uma aula da Asimov Academy.\n    O professor se chama Rodrigo Soares Tadewald.'
```

```
)  
print(transcricao.text)
```

Seja muito bem-vindo ou bem-vinda ao nosso curso completo de Python, aqui da Asimov Academy.

→ Eu e minha equipe ficamos muito felizes que vocês tenham escolhido iniciar no mundo da programação, especificamente com a linguagem Python, aqui com a gente. Pode ter certeza que a gente colocou muito carinho e muita dedicação para construir esse material. Além dos conhecimentos técnicos que a gente vai apresentar sobre a linguagem e programação em si, eu também coloquei grande parte da minha experiência e minha vivência para compartilhar com vocês ao longo desse treinamento. Para quem não me conhece ainda, meu nome é Rodrigo Soares Tadewald e eu não sou programador de origem. Na verdade, eu me formei como engenheiro e utilizei a programação dentro da minha carreira no mercado financeiro como analista de dados. E essa é a grande mágica da programação. Você não precisam utilizá-la única e exclusivamente para desenvolver software. Na verdade, ela pode ser usada para o que vocês quiserem no dia a dia de vocês, muitas vezes como uma habilidade secundária. Ela é muito mais ampla e muito mais poderosa do que isso. A programação é, sem sombra de dúvidas, a habilidade do futuro. Tem gente que já compara e diz que a programação é o novo inglês. Ou seja, até anos atrás, era necessário que a gente tivesse inglês para poder entrar no mercado de trabalho. Hoje, com certeza, além do inglês, a gente tem que saber programar.

E agora o modelo acertou. Ao adicionarmos os nomes próprios no prompt o modelo consegue se corrigir e melhorar sua transcrição. Muito bem.

Podemos também gerar a transcrição no formato de legendas, modificando o response_format para "srt":

```
audio = open('audio_asimov.mp3', 'rb')  
transcricao = client.audio.transcriptions.create(  
    model='whisper-1',  
    file=audio,  
    prompt='Essa é a transcrição de uma aula da Asimov Academy.\n        O professor se chama Rodrigo Soares Tadewald.',  
    response_format='srt'  
)  
print(transcricao)  
  
1  
00:00:01,000 --> 00:00:05,000  
Seja muito bem-vindo ou bem-vinda ao nosso curso completo de Python,  
  
2  
00:00:05,000 --> 00:00:06,500  
aqui da Asimov Academy.  
  
3  
00:00:06,500 --> 00:00:09,500  
Eu e minha equipe ficamos muito felizes que vocês tenham escolhido  
  
4  
00:00:09,500 --> 00:00:11,000  
iniciar no mundo da programação,  
  
5
```

Explorando a API da OpenAI

```
00:00:11,000 --> 00:00:14,000
especificamente com a linguagem Python, aqui com a gente.

6
00:00:14,000 --> 00:00:16,500
Pode ter certeza que a gente colocou muito carinho

7
00:00:16,500 --> 00:00:19,000
e muita dedicação para construir esse material.

8
00:00:19,000 --> 00:00:21,500
Além dos conhecimentos técnicos que a gente vai apresentar

9
00:00:21,500 --> 00:00:24,000
sobre a linguagem e programação em si,

10
00:00:24,000 --> 00:00:27,500
eu também coloquei grande parte da minha experiência e minha vivência

11
00:00:27,500 --> 00:00:30,500
para compartilhar com vocês ao longo desse treinamento.

12
00:00:30,500 --> 00:00:34,000
Para quem não me conhece ainda, meu nome é Rodrigo Soares Tadewald

....
```

E assim facilmente conseguimos criar arquivos de legenda.

20. Mini-projeto - Chatbot com reconhecimento de fala

Para finalizar, criamos um chatbot com reconhecimento de fala que responde de forma falada também.

```
# pip install --upgrade wheel
from io import BytesIO
from pathlib import Path

import speech_recognition as sr
from playsound import playsound

import openai
from dotenv import load_dotenv, find_dotenv

_ = load_dotenv(find_dotenv())
client = openai.Client()

ARQUIVO_AUDIO = 'fala_assistant.mp3'

recognizer = sr.Recognizer()

def grava_audio():
    with sr.Microphone() as source:
        print('Ouvindo...')
        recognizer.adjust_for_ambient_noise(source, duration=1)
        audio = recognizer.listen(source)
    return audio

def transcricao_audio(audio):
    wav_data = BytesIO(audio.get_wav_data())
    wav_data.name = 'audio.wav'
    transcricao = client.audio.transcriptions.create(
        model='whisper-1',
        file=wav_data,
    )
    return transcricao.text

def completa_texto(mensagens):
    resposta = client.chat.completions.create(
        messages=mensagens,
        model='gpt-3.5-turbo-0125',
        max_tokens=1000,
        temperature=0
    )
    return resposta

def cria_audio(texto):
    if Path(ARQUIVO_AUDIO).exists():
        Path(ARQUIVO_AUDIO).unlink()
    resposta = client.audio.speech.create(
        model='tts-1',
        voice='onyx',
```

```
        input=texto
    )
resposta.write_to_file(ARQUIVO_AUDIO)

def roda_audio():
    playsound(ARQUIVO_AUDIO)

if __name__ == '__main__':
    mensagens = []

    while True:
        audio = grava_audio()
        transricao = transricao_audio(audio)
        mensagens.append({'role': 'user', 'content': transricao})
        print(f'User: {mensagens[-1]["content"]}')
        resposta = completa_texto(mensagens)
        mensagens.append({'role': 'assistant', 'content':
→   resposta.choices[0].message.content})
        print(f'Assistant: {mensagens[-1]["content"]}')
        cria_audio(mensagens[-1]["content"])
        roda_audio()
```

21. Finalizando o curso

E assim, chegamos ao fim de mais um curso. Esperamos que este conteúdo tenha sido útil para você! Sinta-se à vontade para compartilhá-lo com seus amigos e, sempre que tiver dúvidas, nos chame nos comentários das aulas que responderemos prontamente!

Um grande abraço!