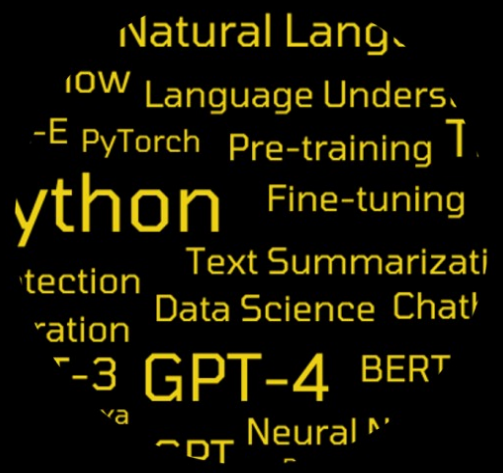


OpenAI GPT

For Python Developers

The art and science of building AI-powered apps with GPT-4, Whisper, Weaviate, and beyond

Suitable for learners of all levels

Aymen EL Amri

www.faun.dev

OpenAI GPT For Python Developers

The art and science of building AI-powered apps with GPT-4, Whisper, Weaviate, and beyond

Published by FAUN.

For permission requests, write to the author at the provided email address: aymen@faun.dev

For general information on our other products and services or to obtain technical support, please contact us at sales@faun.dev

Table of Contents

Preface

When asked about my profession, I often find it challenging to provide a simple answer. My career has taken me through various paths and roles over time. As someone deeply passionate about learning and exploring new things, I've had the opportunity to work in numerous fields.

I've worked in software development, advertising, marketing, network and telecom, systems administration, training, technical writing, computer repair, and other fields. I've always been driven by the desire to learn more and expand my horizons. As I learned new technologies, met new people, and explored new concepts, my mind became more open, and my perspective widened. I began to see connections and possibilities that I had never seen before.

The more I learned, the more I desired to teach, even for free sometimes. I cherish the moment when someone finally understands a concept they've been struggling with. Being a teacher at heart, I've always enjoyed sharing my knowledge.

This is precisely what inspired me to write this guide.

While writing this guide, I constantly considered its future readers. My aim was to create an accessible, easy-to-follow guide on NLP, GPT, and related topics for Python developers with limited knowledge in these areas. My goal was to provide practical information that readers could use to build their own intelligent systems **without needing to spend years learning the theory behind these concepts**.

In this practical, hands-on guide, I share my knowledge and experience with OpenAI's AI models, specifically GPT-3 and GPT-4, as well as other models. I explain how Python developers can use them to add intelligence to their new or existing applications.

This guide is designed as a step-by-step manual that covers the fundamental concepts and techniques of using AI/ML and OpenAI models, providing a hands-on approach to learning.

My inbox is always full, as I receive numerous emails. However, the most rewarding ones are from readers who have found my online guides and courses useful. Please feel free to reach out to me at aymen@faun.dev and share your testimonial - I would love to hear from you.

I hope you enjoy reading this guide as much as I enjoyed writing it.

About the Author

Aymen El Amri is an author, entrepreneur, trainer, and polymath software engineer. He has worked in various roles, responsibilities, and projects in the technology field, including DevOps & Cloud Native, Cloud Architecture, Python, NLP, Data Science, and more.

Aymen has trained thousands of software engineers and has written multiple books and courses read by thousands of other developers and software engineers.

His teaching approach is practical, based on simplifying complex concepts into easy-to-understand language and providing real-world examples that resonate with his audience.

He founded projects such as FAUN, eralabs, and Marketto. You can find Aymen on Twitter and Linkedin.

The Story of OpenAI and ChatGPT

In December 2015, a group of brilliant minds united with a shared goal: to foster and develop friendly AI for the benefit of all humanity.

Sam Altman, Elon Musk, Greg Brockman, Reid Hoffman, Jessica Livingston, Peter Thiel, Amazon Web Services (AWS), Infosys, and YC Research announced the establishment of OpenAI, pledging over US$1 billion to the initiative. The organization declared its intention to "freely collaborate" with other institutions and researchers by making its patents and research publicly available.

OpenAI's headquarters are located in the Pioneer Building in San Francisco's Mission District. In April 2016, OpenAI launched a public beta of "OpenAI Gym", a platform for reinforcement learning research. In December 2016, OpenAI unveiled "Universe", a software platform designed to measure and train an AI's general intelligence across a wide array of games, websites, and other applications.

In 2018, Elon Musk stepped down from his board seat, citing "a potential future conflict of interest" with Tesla's AI development for self-driving cars, but continued to support the organization as a donor. In 2019, OpenAI transitioned from a non-profit to a capped-profit entity, with a profit cap set at 100 times any investment. The company distributed equity to its employees and formed a partnership with Microsoft, which announced an investment of US$1 billion in the company. OpenAI subsequently announced plans to commercially license its technologies.

In 2020, OpenAI unveiled GPT-3, a language model trained on trillions of words from the internet. They also announced that an associated API, simply named "the API", would form the basis of its first commercial product. GPT-3 is designed to answer questions in natural language, but it can also translate between languages and generate coherent improvised text.

In 2021, OpenAI introduced DALL-E, a deep-learning model capable of generating digital images from natural language descriptions.

Fast forward to December 2022, OpenAI received widespread media coverage after launching a free preview of ChatGPT. According to OpenAI, the preview attracted over a million signups within the first five days. Anonymous sources cited by Reuters in December 2022 projected OpenAI's revenue at US$200 million for 2023 and US$1 billion for 2024. As of January 2023, the company was in discussions for funding that would value it at $29 billion.

This is the brief history of OpenAI, an artificial intelligence research laboratory composed of the for-profit OpenAI LP and its parent company, the non-profit OpenAI Inc.

Most people were unaware of OpenAI until the company released its popular ChatGPT.

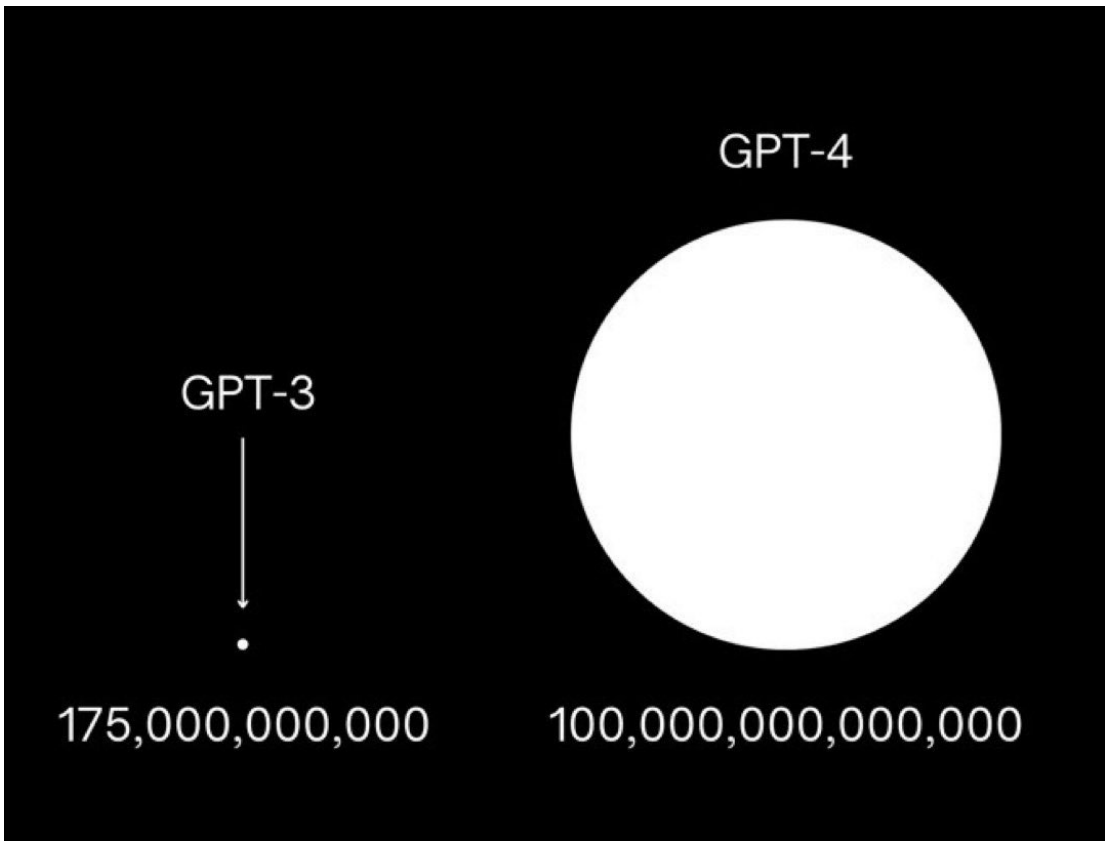
The primary purpose of ChatGPT was to mimic human behavior and engage in natural conversations with people. However, the chatbot can learn and teach itself based on the conversations it has with different users. This AI has conversational capabilities and can write tutorials and code, compose music, and perform other tasks. The use cases for ChatGPT are quite diverse and potentially limitless, as users have demonstrated. Some use cases are creative (e.g., writing a rap song), others are malicious (e.g., generating malicious code or commands), and still, others are business-oriented (e.g., SEO, content marketing, email marketing, cold emails, and business productivity).

ChatGPT, standing for **G**enerative **P**re-trained **T**ransformer, is built on top of OpenAI's GPT family of large language models (LLMs). The chatbot is fine-tuned using both supervised and reinforcement learning techniques.

GPT-3 served as the foundation for ChatGPT before the release of GPT-3.5 on March 15, 2022, followed by GPT-4 on March 14, 2023. ChatGPT is a project that utilizes GPT-3.5 and GPT-4, and adds a web interface, memory, and more user-friendly features. After reading this guide, you will be

capable of building your own chatbot, potentially better than ChatGPT, since you can customize it to your specific needs.

You may have seen the chart comparing the two versions of GPT, showing the number of parameters in GPT-3 (175 billion) and GPT-4 (100 trillion).



GPT-3 vs 4

Altman referred to this viral illustration as "*complete nonsense*."

> "*The GPT-4 rumor mill is a ridiculous thing. I don't know where it all comes from. People are setting themselves up for disappointment, and they will be. The hype is just like… We don't have an actual AGI, and that's sort of what's expected of us.*"

The internet is rife with rumors, speculations, and low-quality content. This is true even in the AI world. You might have come across numerous

tutorials and books promising to teach you how to use OpenAI GPT models, only to find a collection of copy-pasted prompts from the internet. This guide is different. Learning how to use OpenAI models isn't about having a cheat sheet of prompts; it's about understanding the principles and techniques behind these models, how to utilize them in building your applications, and how they can integrate with your existing systems and other AI/ML tools.

Currently, many projects are leveraging OpenAI models as the backbone for their products. Some of the most notable ones include:

- GitHub Copilot: This uses the OpenAI Codex model, a derivative of GPT-3, fine-tuned for generating code.
- Copy.ai and Jasper.ai: These are used for content generation for marketing purposes.
- Drexel University: This is used for the detection of early signs of Alzheimer's disease.
- Algolia: This is used to enhance their search engine capabilities.
- Your next AI startup?

By following this guide, you'll learn how to effectively use OpenAI models in your projects, and you might be the next one to join this list.

About This Guide

The knowledge you'll acquire from this guide will be applicable to the current families of GPT models (GPT-3, GPT-3.5, GPT-4, etc.) and will likely also be relevant to GPT-5, should it ever be released.

OpenAI provides APIs (Application Programming Interfaces) to access their AI. The goal of an API is to abstract the underlying models by creating a universal interface for all versions, allowing users to use GPT regardless of its version.

This guide aims to provide a comprehensive, step-by-step tutorial on how to utilize GPT-3.5 and GPT-4 in your projects via this API. It also covers other models, such as Whisper and Text-to-Speech.

If you're developing a chatbot, an AI assistant, or a web application that utilizes AI-generated data, this guide will assist you in achieving your objectives.

If you have a basic understanding of the Python programming language and are willing to learn a few additional techniques, such as using Pandas Dataframes and some NLP methods, you possess all the necessary tools to start building intelligent systems with OpenAI tools.

Rest assured, **you don't need to be a data scientist, machine learning engineer, or AI expert** to comprehend and implement the concepts, techniques, and tutorials presented in this guide. The explanations provided are straightforward and easy to understand, featuring simple Python code, examples, and hands-on exercises.

This guide emphasizes practical, hands-on learning and is designed to assist readers in building real-world applications. It is example-driven and provides numerous practical examples to help readers understand the concepts and apply them to real-life scenarios to solve real-world problems.

By the end of your learning journey, you will have developed applications such as:

- Fine-tuned, domain-specific chatbots.
- An intelligent conversational system with memory and context.
- A semantic modern search engine using RAG and other techniques.
- An intelligent coffee recommendation system based on your taste.
- A chatbot assistant to assist with Linux commands.
- A fine-tuned news category prediction system.
- An AI-to-AI autonomous discussion system to simulate human-like conversations or solve problems.
- An AI-based mental health coach trained on a large dataset of mental health conversations.
- and more!

By reading this guide and following the examples, you will be able to:

- Understand the different models available, and how and when to use each one.
- Generate human-like text for various purposes, such as answering questions, creating content, and other creative uses.
- Control the creativity of GPT models and adopt the best practices to generate high-quality text.
- Transform and edit the text to perform translation, formatting, and other useful tasks.
- Optimize the performance of GPT models using various parameters and options such as `max_tokens`, `temperature`, `top_p`, `n`, `stream`, `logprobs`, `stop`, `presence_penalty`, `frequency_penalty`, `best_of`, and others.
- Stem, lemmatize, and reduce your costs when using the API.
- Understand Context Stuffing, chaining, and practice prompt engineering.
- Implement a chatbot with memory and context.
- Create prediction algorithms and zero-shot techniques and evaluate their accuracy.
- Understand, practice, and improve few-shot learning.
- Understand fine-tuning and leverage its power to create your own fine-tuned models.
- Understand and use fine-tuning best practices.
- Practice training and classification techniques using GPT.
- Understand embedding and how companies such as Tesla and Notion are using it.
- Understand and implement semantic search, RAG and other advanced tools and concepts.
- Integrate a Vector Database (e.g.: Weaviate) with your intelligent systems.

The Companion Toolkit

For an enhanced learning experience, we have created a companion toolkit that includes all the code snippets, files, and datasets used in this guide. You can download it using the following URL: from.faun.to/r/AyEg.

Stay Connected

If you are interested in staying updated with the rapid advancements in the Python and AI ecosystems, I warmly invite you to join our vibrant developer community at www.faun.dev/join. As a member of our community, you will receive weekly newsletters designed to keep you informed and ahead of the curve. These newsletters are meticulously curated, filled with must-read tutorials, the latest news, and profound insights from leading experts in the software engineering community. By leveraging this resource, you will ensure that you are always up-to-date with the latest trends and developments in the Python and AI fields, giving you an advantage in your technological journey.

How Does GPT Work?

The Generative Pre-trained Transformer, or GPT, is a generative text model. This model can produce new text by predicting what comes next based on the input it receives.

GPT-4 is another model that is larger and more performant than any previous GPT model, including GPT-1/2/3/3.5.

The 4th generation was trained on a large corpus of text, such as books, articles, and publicly accessible websites like Reddit and other forums. It uses this training data to learn patterns and relationships between words and phrases.

GPT-4's key innovation lies in its impressive size - boasting billions of parameters - making it one of the most massive and powerful language models ever devised. Its extensive training on such a vast dataset enables it to generate human-like text, execute various natural language processing tasks, and complete tasks with impressive accuracy.

GPT is a type of neural network known as a transformer, which is specifically designed for natural language processing tasks. The architecture of a transformer is based on a series of self-attention mechanisms that allow the model to process input text in parallel and weigh the importance of each word or token based on its context.

Self-attention is a mechanism used in deep learning models for natural language processing (NLP). It allows a model to weigh the importance of different parts of a sentence or multiple sentences when making predictions. As part of the Transformer architecture, it enables a neural network to achieve satisfactory performance in NLP tasks. The concept of self-attention is based on the idea that each word in a sentence can be influenced by other words in the sentence, and the importance of each word can be determined based on its context. The original work on self-attention was introduced in the paper "Attention is All You Need".

To better illustrate the concept of self-attention, consider the following analogy:

Imagine attending a large dinner party where everyone is sharing stories simultaneously. You're trying to follow all the conversations around you, but naturally, you focus more on the stories that are relevant to you or contain important information. As people speak, you give your attention to the storyteller whose narrative is currently most interesting or relevant, sometimes switching focus based on new information or connections you make to other stories being told.

In this analogy, you are akin to the transformer model in GPT. The dinner party symbolizes the input text, where each guest's story represents a word or token in the sequence. Your capacity to listen to multiple stories simultaneously and decide which one to focus on is similar to the self-attention mechanism. This mechanism enables the model to process all parts of the input at the same time and determine the relevance or importance of each word in the context of all the others, much like you tuning into the most engaging story at any given moment.

Let's examine a quick code example. This is an instance of using Hugging Face Transformers for GPT-2 inference:

```
from transformers import pipeline
generator = pipeline(
    'text-generation',
    model = 'gpt2'
)
generator(
    "Hello, I'm a language model",
    max_length = 30,
    num_return_sequences=3
)
```

This is an example of an output from the code above:

```
[
    {
        'generated_text':
        "Hello, I'm a language model. So while writing this, ..."
    },
```

```
    {
        'generated_text':
        "Hello, I'm a language model. I write and maintain ..."
    }
]
```

By default, a model has no memory, meaning that each input is processed independently, without any information being carried over from previous inputs. When GPT generates text, it doesn't have any preconceived notions about what should come next based on previous inputs. Instead, it generates each word based on the probability of it being the next likely word given the previous input. This results in text that can be surprising and creative.

Here is another example of code that uses a GPT model to generate text based on user input.

```
# Import the necessary libraries
from transformers import GPT2Tokenizer, GPT2LMHeadModel

# Load the pre-trained GPT-2 tokenizer and model
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2')

# Set the model to evaluation mode
model.eval()

# Define a prompt for the model to complete
prompt = input("You: ")

# Tokenize the prompt and generate text
input_ids = tokenizer.encode(prompt, return_tensors='pt')
output = model.generate(input_ids, max_length=50, do_sample=True)

# Decode the generated text and print it to the console
generated_text = tokenizer.decode(output[0], skip_special_tokens=True)
print("AI: " + generated_text)
```

GPT-4 was designed as a general-purpose large multimodal model, which means that it can be used for a wide variety of natural language processing tasks (such as language translation, text summarization, and question answering) and multimodal tasks (such as text-to-image generation and image captioning). OpenAI has trained GPT-4, but you can base your own model on your fine-tuned datasets. This allows for more creative and

specific use cases, in addition to the default tasks the model can assist with, such as generating text, poetry, and stories. It can also be used for building chatbots that are experts in a specific domain, other conversational interfaces, and more!

In this guide, we will dive deeply into OpenAI models (GPT, embeddings, and more) and how to use them in Python, including how to effectively utilize the API and the tools provided by the company and the AI/ML community to prototype powerful and creative tools and systems. This will not only enhance your proficiency in using the GPT API but also broaden your understanding of the concepts and techniques used in natural language processing, AI/ML, and other related fields.

GPT is regarded as a significant advancement in natural language processing due to its extensive scale. However, some experts have expressed concerns about the potential for the model to generate biased or harmful content. Like any technology, it’s extremely important to pause and consider the ethical implications of its use. This guide will not address the ethical issues but will solely focus on the practical technical aspects.

Happy coding!

Setting Up the Development Environment

Notes

This guide frequently employs the “heredoc” syntax for creating files and inserting content directly from the command line or within scripts and programs.

> **i** Heredoc, short for “here document,” is a feature of shell scripting that simplifies the creation of multiline strings.

The heredoc syntax begins with << followed by a delimiter token of your choice, continues with the content you wish to include, and ends with the same delimiter token on a new line. This syntax is particularly useful for scripting and programming tasks that necessitate file generation or multiple lines of input for a command.

To utilize the heredoc syntax for file creation or content appending, follow the structure below:

```
cat << EOF > filename
This is line #1
This is line #2
..
EOF
```

In this example, `<<EOF` denotes the commencement of the heredoc section, and `EOF` on a separate line signifies its conclusion. The content between these markers is redirected into “filename”. Replace “filename” with your preferred file name and adjust the content as needed.

The primary reason for using the heredoc syntax in this guide is to allow readers to create files accurately and effortlessly with a single copy-paste action. This method ensures that the author's exact content is duplicated in the reader's environment, minimizing the chance of manual errors and enhancing the learning experience.

I recommend copying and pasting the heredoc examples directly into your terminal to create the files and content as intended.

Windows users can execute the heredoc examples using the Windows Subsystem for Linux (WSL). Alternatively, you can manually create the files and paste the content using a text editor. However, I suggest using a Unix-like environment (preferably a Debian-based system) to follow the examples in this guide.

If you're using a different system, you may need to modify some minor details, such as the package manager, the installation process for packages, and the method for creating files and directories. This should not be an issue, as the examples are straightforward and can be easily adapted to any system.

Another important note worth mentioning is that this guide is structured in a way where each chapter builds upon the previous one. Topics and concepts are interwoven throughout to form a comprehensive understanding of the subject matter. It's crucial for you to progress through the chapters sequentially, fully grasping each section before moving on to the next. Patience and attention to detail will greatly enhance your learning experience, allowing you to fully benefit from the structured progression.

The code presented in this guide is not meant to be passively read but actively executed. Engaging with the code - running it, modifying it, and observing the outcomes—is crucial for a deep understanding of the concepts discussed. This hands-on approach will help you internalize the material and develop the skills necessary to apply the concepts in your own projects.

Installing Python, pip, and a Virtual Development Environment

First and foremost, you'll need Python. For the purposes of this guide, we'll be using Python 3.9.5.

We'll also be using pip, which is a package installer for Python. Pip allows you to install packages from the Python Package Index and other indexes.

Don't worry if you don't have Python 3.9.5 installed on your system. We're going to create a virtual development environment, so regardless of the version you currently have, your development environment will be isolated from your system and will use Python 3.9.5.

If Python isn't already installed on your system, head over to www.python.org/downloads/, download, and install one of the Python 3.x versions. The instructions you'll need to follow will vary depending on your operating system.

To manage our development environment, we'll be using virtualenvwrapper. You can find the installation instructions in the official documentation.

```
pip install --upgrade pip
pip install virtualenv
pip install virtualenvwrapper
```

For Windows users, the following command can be used to install virtualenvwrapper-win:

```
pip install --upgrade pip
pip install virtualenv
pip install virtualenvwrapper-win
```

Note: If you're accustomed to using virtualenv, Poetry, Conda, or any other environment, feel free to continue using it. There's no need to install

virtualenvwrapper in this case.

If pip isn't installed, the simplest way to install it is by using the "get-pip.py" script:

```
curl https://bootstrap.pypa.io/get-pip.py | python3
```

Windows users can use the following command:

```
curl https://bootstrap.pypa.io/get-pip.py
py get-pip.py
```

In summary, your system should have the following packages installed:

- Python 3.9
- Pip
- Virtualenv
- virtualenvwrapper (optional, as you can use virtualenv or Poetry instead)

Again, I strongly recommend Windows users to create a Debian-based virtual machine, as the examples provided in this guide were tested on a Debian-based system. While most of the commands and code snippets will work on Windows, some may need to be adapted.

Next, we'll create a virtual environment:

```
mkvirtualenv -p python3.9 openaigptforpythondevelopers
```

Once it's created, activate it:

```
workon openaigptforpythondevelopers
```

Obtain Your OpenAI API Keys

The next step involves generating API keys that will grant you access to the official API provided by OpenAI.

Visit platform.openai.com to create an account, then generate your API keys.

An API key can be associated with an organization, and you'll be prompted to create one. In this guide, we'll name it: "**LearningGPT**".

Ensure to store the generated secret key in a safe and accessible location. You won't be able to retrieve it again through your OpenAI account.

Install the Official Python Bindings

While you can interact with the API using HTTP requests from any language, this can be done either via the official Python bindings, the official Node.js library, or a community-maintained library.

In this guide, we will utilize the official library provided by OpenAI. Another option is Chronology, an unofficial library offered by OthersideAI. However, this library appears to be no longer maintained, and its use in production is not recommended.

To install the official Python bindings, execute the following command:

```
pip install openai==1.9.0
```

Ensure that you are installing the library in the virtual environment we created earlier.

Test our API Keys

First, let's set the API key and the organization ID as environment variables:

```
export API_KEY=xxx
export ORG_ID=xxx
```

Now, you can run the following command to test the API:

```
curl \
  https://api.openai.com/v1/models \
  -H 'Authorization: Bearer '$API_KEY'' \
  -H 'OpenAI-Organization: '$ORG_ID''
```

If you only have one organization in your OpenAI account, you can run the same command without specifying the organization ID.

```
curl \
  https://api.openai.com/v1/models \
  -H 'Authorization: Bearer '$API_KEY''
```

The curl command should return a list of models provided by the API, such as "davinci", "ada", and many others.

The second test we will conduct involves using the Python SDK.

Start by creating a file named ".env", where you will store the API key and organization ID:

```
cat << EOF > .env
API_KEY=$API_KEY
ORG_ID=$ORG_ID
EOF
```

To test the API using the Python SDK, execute the following code:

```
cat << EOF > test_api.py
import os
from openai import OpenAI
from pprint import pprint

# reading variables from .env file,
# namely API_KEY and ORG_ID.
with open(".env") as env:
    for line in env:
        key, value = line.strip().split("=")
        os.environ[key] = value

# Initializing the API key and organization id
client = OpenAI(
  api_key=os.environ['API_KEY'],
  organization=os.environ['ORG_ID']
)

# Printing the client object
pprint(vars(client))
EOF
```

Run the above code by executing the following command:

```
python test_api.py
```

We will authenticate using the API key and organization ID in each code snippet throughout this guide. That's why we are organizing our code in the following manner:

Create a folder named "src" and inside it, create a file named "api.py":

```
cat << EOF > src/api.py
import os
from openai import OpenAI

# reading variables from .env file,
# namely API_KEY and ORG_ID.
with open("src/.env") as env:
    for line in env:
        key, value = line.strip().split("=")
        os.environ[key] = value
```

```
# Instantiating the client at module level
client = OpenAI(
  api_key=os.environ['API_KEY'],
  organization=os.environ['ORG_ID']
)
EOF
```

Set up the ".env" file as follows:

```
cat << EOF > src/.env
API_KEY=$API_KEY
ORG_ID=$ORG_ID
EOF
```

To test the API, execute the following code:

```
cat << EOF > src/test_api.py
from api import client
from pprint import pprint

# Printing the client object
pprint(vars(client))
EOF
```

Run the above code by executing the following command:

```
python src/test_api.py
```

We will use this same structure for the rest of the code snippets in this guide.

Understanding the Available Models and Which One to Use

OpenAI Available Models and Important Considerations

Using the API's models endpoint, you can list all available models. Let's see how this works in practice:

```
cat << EOF > src/test_api.py
from api import client

models = client.models.list()
for model in models:
    print(vars(model))
EOF
```

Run the code:

```
python src/test_api.py
```

You should see a similar output to this:

```
{
    'id': 'curie-search-query',
    'created': 1651172509,
    'object': 'model',
    'owned_by': 'openai-dev'
}

{
    'id': 'babbage-search-query',
    'created': 1651172509,
    'object': 'model',
    'owned_by': 'openai-dev'
}

    {
    'id': 'dall-e-3',
    'created': 1698785189,
    'object': 'model',
    'owned_by': 'system'
```

```
}
... etc
```

The OpenAI Python SDK is intuitive and easy to use. We simply use the `list()` method to list all the available models.

Let's print just the IDs of the models:

```
cat << EOF > src/test_api.py
from api import client

models = client.models.list()
for model in models:
    print(model.id)
EOF
```

Run the code:

```
python src/test_api.py
```

The result should contain a list of all the available models; these are the most common ones:

```
dall-e-3
dall-e-2
GPT-4
davinci-002
babbage-002
whisper-1
GPT-3.5-Turbo-16k
GPT-3.5-Turbo
GPT-3.5-Turbo-Instruct
```

The models Babbage-002 and Davinci-002 are both part of the GPT-3 series developed by OpenAI, and they serve as replacements or upgrades to previous models:

- Babbage-002: Serves as a replacement for the GPT-3 Ada and Babbage base models. It's optimized for tasks that require a balance between performance and cost-efficiency and is suitable for moderately complex tasks like text classification, semantic search, and basic language understanding.
- Davinci-002: Replaces the GPT-3 Curie and Davinci base models. It's known for its advanced language understanding and generation capabilities and is ideal for more complex tasks that require nuanced understanding, such as

content creation, complex language translation, and solving sophisticated problems.
- DALL-E 3 and DALL-E 2: Both are image generation models. DALL-E 3 is an improvement over DALL-E 2 - a basic model, with more advanced capabilities in generating high-resolution, realistic images from text descriptions.
- Whisper-1: This model is specialized in speech recognition and transcription. It was designed to convert spoken language into written text accurately. In the future, you may see more advanced versions of this model (e.g., Whisper-2, Whisper-3, etc.).
- GPT-3.5-Turbo and GPT-3.5-Turbo-Instruct: All of these models are variants of GPT-3.5 designed for more efficient and responsive text generation.
- GPT-4: This is a large multimodal model that can accept both text and image inputs. It is the successor to GPT-3 and is capable of performing a wide range of tasks.

Some of the models listed in the output are dynamic, as they are subject to continuous updates by OpenAI. For example, GPT-3.5-Turbo, GPT-4, and GPT-4-32k point to the latest model version. At the time of writing, the latest GPT-4 model is GPT-4-0613, so when your code calls the GPT-4 API endpoint, you are actually using GPT-4-0613. In the future, when OpenAI releases a new GPT-4 model, the API endpoint will point to the new model. This is why it is recommended to use a specific version of the model in your code if you have specific requirements.

Historically, models were called “engines”, but OpenAI deprecated the term “engine” in favor of “model”. Some resources use these terms interchangeably, but the correct term is “model”. When using the API, requests that use the old names will still work, as OpenAI has ensured backward compatibility, but this could change in the future. However, some models should be manually replaced by newer ones. It is recommended that you use the updated models to avoid any issues in the future.

Some models are not listed above, like the moderation models. These models are used to detect inappropriate content in text and to filter it out. The usage of these models is optional but highly recommended if you are building a public application. You don’t want your users to receive inappropriate results.

Which Model to Use?

There is no single answer to this question. It depends on your use case and the type of application you are building. In general, you should use the most recent model available. For example, from a functionality perspective, if you are building a chatbot, you should use the GPT-3.5-Turbo-Instruct model. If you are building an application that generates images from text, you should use the DALL-E 3 model. If your application requires more capabilities, you should use the GPT-4 model.

The second factor to consider is pricing. Some models are more expensive than others. For example, GPT-3.5-Turbo is the most cost-effective model in the GPT-3.5 series.

Another important factor is the context window. Some models have a larger context window than others. For example, the GPT-4-32k model has a context window of 32,768 tokens (about 50 pages of text), while the GPT-4-0613 model has a context window of 8,192 tokens (about 12 pages of text). This means that the GPT-4-32k model can process more text than the GPT-4-0613 model.

In the following sections, we're going to take a closer look at the main models available. This will help you understand the differences between them and choose the right model for your application. However, it's important to understand that some of the models listed below are deprecated and likely won't be available in the future. OpenAI is constantly improving its models and deprecating old ones.

OpenAI Model Series

GPT-4 Series

GPT-4 is the latest model series developed by OpenAI and is the successor to GPT-3. One of the main differences from GPT-3 is that GPT-4 models are multimodal. This means that GPT-4 can accept both image and text inputs.

> **i** In general, multimodal deep learning is a subfield of deep learning that enriches models with information collected from various data modalities, such as text, images, video, audio, and sensor data. This puts it a step ahead of traditional machine learning, which was confined to a single modality, thus addressing the complexity surrounding real-world data that often blends various sources. The objective is to create a unified representation space,