

Introduction to ChatGPT, Whisper & Github Copilot for Python Development
by Martin Brochhaus

About PUGS

- PUGS stands for **Python User Group Singapore**
 - registered non-profit society, run by volunteers
 - was created to organize PyCon Singapore
- Visit <https://pugs.org.sg/membership/> to become a member
- Monthly meetups at <https://www.meetup.com/singapore-python-user-group/>

The screenshot shows the PyCon SG 2019 website. The main header reads "PYCON SINGAPORE 2019 OCT 10-12 BY THE COMMUNITY | FOR THE COMMUNITY". Below the header is a stylized illustration of the Singapore skyline featuring the Merlion and various landmarks. A large white box at the bottom left contains the text "Ticket & Tutorial Sales Are Closed" and "PyCon Singapore 2019 is happening from Thursday to Saturday on October 10 - October 12 2019 at Republic Polytechnic!". A note below states: "We would like to thank all our sponsors, speakers and attendees and we are looking forward to an exciting conference. Everyone should have gotten a welcome email that describes how to find the venue and answers other common questions. If you didn't get this email, please message conference@pycon.sg and we will forward you a copy."

This image shows a mobile-style interface for selecting a membership tier. It features three options: "Free", "SGD 0", and "SGD 50/year". Each option includes a brief description and a "Choose" button at the bottom.

- Free**: Access to our monthly newsletter
- SGD 0**:
 - Your support helps us to grow a sustainable and vibrant community
 - Access to our monthly newsletter
 - Official member of the Python User Group Singapore Society with voting rights to the annual AGM
 - Access to discounted PyCon SG tickets
 - Access to members-only networking events
- SGD 50/year**:
 - Your support greatly helps us to organise bigger and better PyCon SG conferences
 - Your logo will appear on our monthly newsletter
 - You may announce Python related job offers from your company in our monthly newsletter
 - You will receive a Community Sponsor badge at PyCon SG events and on our official website with your logo and a short company description
 - Once you subscribe to this tier, we will reach out to you and ask for your company logo and description

The screenshot shows the Meetup group page for the "Singapore Python User Group". The page includes a photo of a group event, basic group stats (4.5 stars, 6,090 members), and sections for "About", "Events", "Members", "Photos", "Discussions", and "More". The "About" section describes the group as being for programmers from all walks of Python life. The "Upcoming events" section lists an event titled "Introduction to ChatGPT & Github Copilot for Python" on Fri, May 5, 2023, at 6:30 PM SGT.

About me

- Martin Brochhaus
- CTO of theartling.com
- Committee member of PUGS
- Twitter: [@mbrochh](https://twitter.com/mbrochh)

The screenshot shows the homepage of theartling.com. At the top, there's a navigation bar with links for Art, Design, Discover, Collections, and Art Consultancy. A search bar with the placeholder "What are you looking for?" is positioned above a grid of artwork thumbnails. Below the search bar are several filters: Price (0-10000), Orientation (horizontal/vertical), Color (a palette of colors), Artist From (Any Country), Ships From (Any Item Location), Medium (Any Medium), Material (Any Material), Artwork Style (Any Style), and Size (cm/inches). The main content area displays a grid of four artworks. The first artwork, "War v.3" by Inna Mosina from Kazakhstan, is a photograph of a man's face suspended by strings. The second, "Drawing #05" by Sunit Mehdiratta from India, is a drawing of a figure. The third, "Vibrate" by Paula Callejo from Ecuador, is a painting of colorful, flowing lines. The fourth, "Circuito #16 A-B (Diptych)" by Alec Franco from Argentina, is a painting of abstract blue and black shapes.

This is a screenshot of Martin Brochhaus' Twitter profile (@mbrochh). The profile picture is a portrait of him wearing glasses. The header image is a night photograph of a city skyline with many lit-up skyscrapers. On the left, there's a sidebar with icons for Home, Trends, Notifications, Direct Messages, Bookmarks, and Profile. The bio reads: "I post about Bitcoin, Books, React, Django, Nutrition. Probably in that order. Full Stack developer since 1998." Below the bio, it says "Singapore" with a location pin icon, "Born September 8, 1982" with a birthday icon, and "Joined March 2009" with a calendar icon. At the bottom, it shows "10 Following" and "1,880 Followers".

Say Hi to GPT-4!

- GPT stands for **G**enerative **P**re-trained **T**ransformer
- It is a multimodal large language model
 - this means it can also take images as input
- It has a context window of 8192 and 32768 tokens
 - depending on how much you are willing to pay
- It was released on March 14, 2023
- Sam Altman said it has cost \$100 million to train the model
- There are rumors that it has 1 trillion parameters

Say Hi to GPT-4!

OpenAI's "GPT-n" series

Model	Architecture	Parameter count	Training data	Release date	Training cost
GPT-1	12-level, 12-headed Transformer decoder (no encoder), followed by linear-softmax.	117 million	BookCorpus: ^[23] 4.5 GB of text, from 7000 unpublished books of various genres.	June 11, 2018 ^[5]	"1 month on 8 GPUs" ^[5] , or 1.7e19 FLOP. ^[24]
GPT-2	GPT-1, but with modified normalization	1.5 billion	WebText: 40 GB of text, 8 million documents, from 45 million webpages upvoted on Reddit.	February 14, 2019 (initial/limited version) and November 5, 2019 (full version) ^[25]	"tens of petaflop/s-day" ^[26] , or 1.5e21 FLOP. ^[24]
GPT-3	GPT-2, but with modification to allow larger scaling	175 billion	570 GB plaintext, 0.4 trillion tokens. Mostly CommonCrawl, WebText, English Wikipedia, and two books corpora (Books1 and Books2).	May 28, 2020 ^[26] (then March 15, 2022, for a revision ultimately termed GPT-3.5)	3630 petaflop/s-day (Figure 2.2 ^[26]), or 3.1e23 FLOP. ^[24]
GPT-4	Also trained with both text prediction and RLHF; accepts both text and images as input. Further details are not public. ^[22]	Undisclosed	Undisclosed	March 14, 2023	Undisclosed. Estimated 2.1e25 FLOP. ^[24]

Source: *Wikipedia*

The rise of ChatGPT

- It took ChatGPT only two months to reach 100 million monthly active users



Source: [How ChatGPT Works Technically | ChatGPT Architecture](#)

Attention Is All You Need

- 2017 Paper started it all: <https://arxiv.org/pdf/1706.03762.pdf>



Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Lukasz Kaiser*
Google Brain
lukasz Kaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

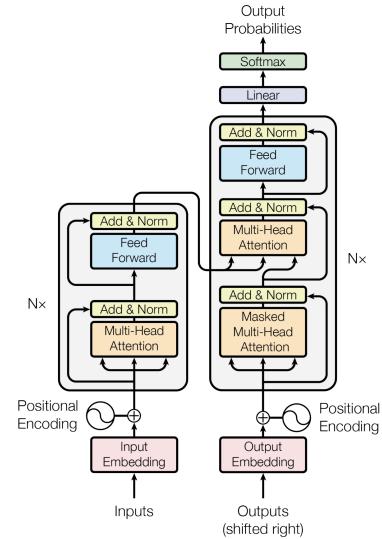


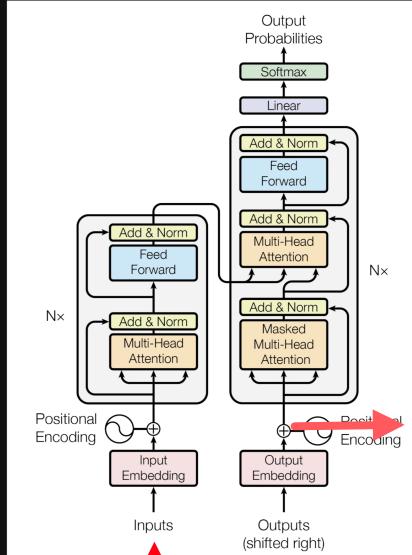
Figure 1: The Transformer - model architecture.

1 Trillion Parameters?

- The goal with any machine learning model is to create a function
- We want to throw some input `x` into the function
- The function shall return a predicted value `y` to us
- For language models, `x` might be a word or sentence
- As a first step, that text will get converted into numbers so that we can do math with it
- The function will then spit out another number, which stands for another word that is most likely to come next
- Our function might look something like this: $f(x) = ax + bx + cx + dx + ex + fx$
- But: it doesn't stop at `f`, it has A LOT more unknown parameters
- GPT-3 had 175 billion parameters, GPT-4 is rumoured to have 1 trillion parameters

How does it train?

- First, we give it some text as input

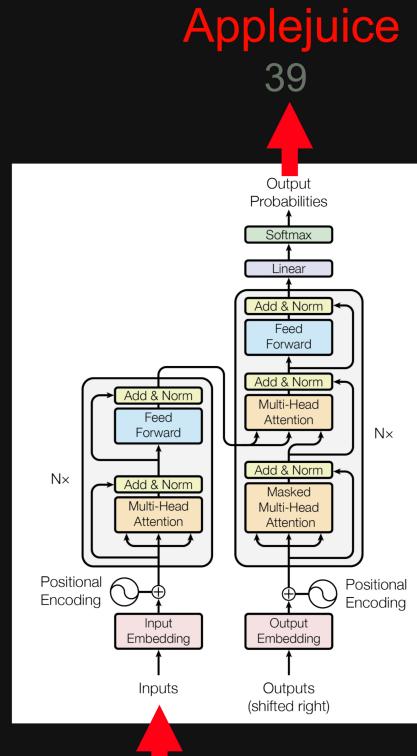


"May the force be"

12 1 28 5

How does it train?

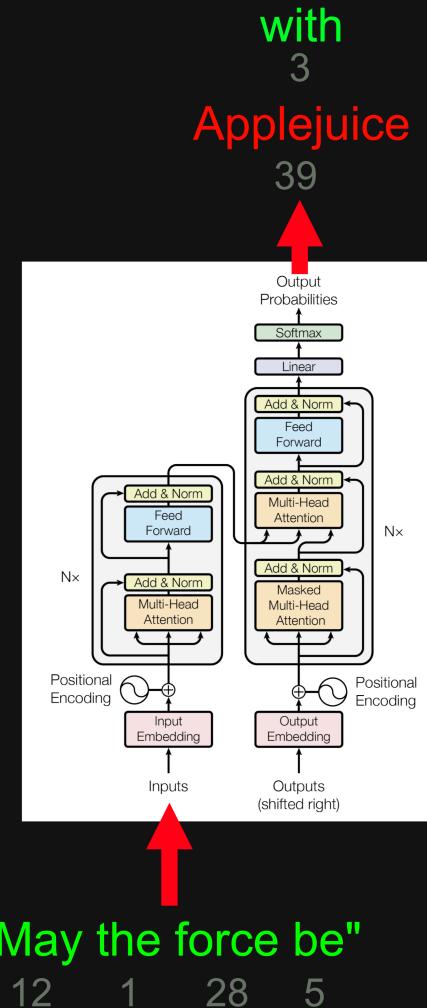
- First, we give it some text as input
- It will then "predict" some random nonsense



"May the force be"
12 1 28 5

How does it train?

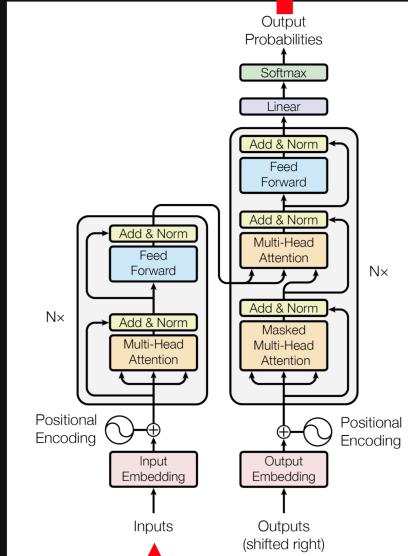
- First, we give it some text as input
- It will then "predict" some random nonsense
- We then reveal the correct answer



How does it train?

- First, we give it some text as input
- It will then "predict" some random nonsense
- We then reveal the correct answer
- We calculate the error, or "loss" between the prediction and the correct answer

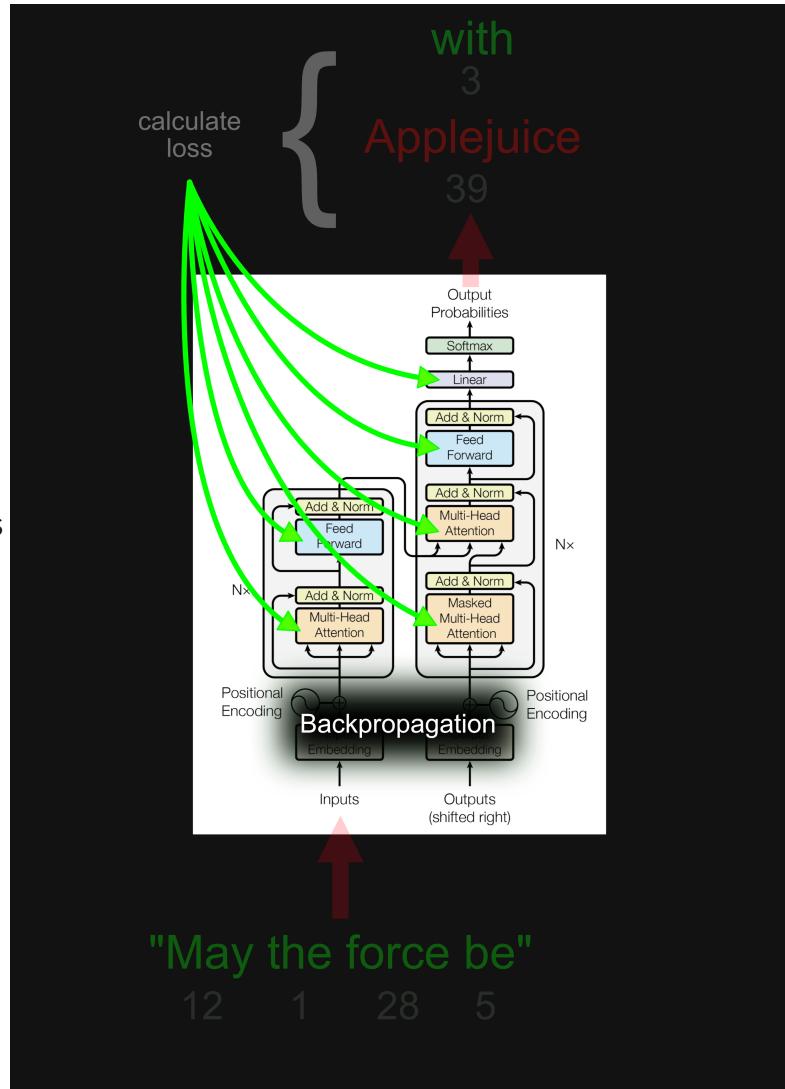
calculate loss
with
3
Applejuice
39



"May the force be"
12 1 28 5

How does it train?

- First, we give it some text as input
- It will then "predict" some random nonsense
- We then reveal the correct answer
- We calculate the error, or "loss" between the prediction and the correct answer
- We use that loss value to update all the parameters in the model a tiny bit via a process called "backpropagation"
- We repeat that billions of times with billions of different inputs



So what are "tokens"?

- It would be inefficient to use all words of all languages as our vocabulary
- A so called `Byte Pair Encoding` (BPE) algorithm can create a new, smaller vocabulary
- GPT-4 can keep 8192 or 32768 tokens in mind

- astrology
- astronomist
- astronomy

- --> astro, no, my, logy, mist

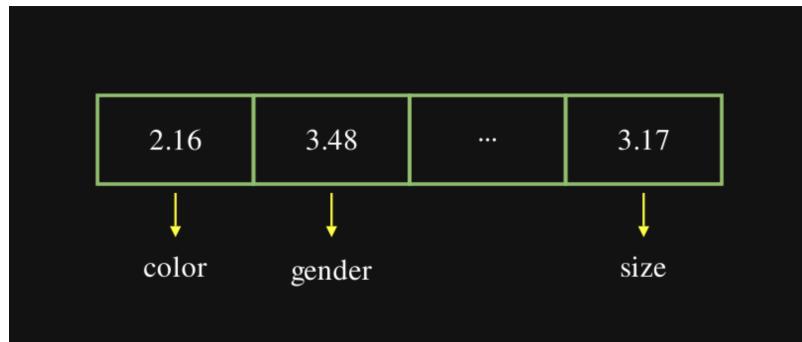
Converting tokens into vectors

- Computers don't understand words, they need numbers
- During the process of training, the model comes up with a vector for every word in the vocabulary
- The `text-embedding-ada-002` model uses a vector with 1536 dimensions

May	-4.97	-1.05	...	0.2
the	-3.03	1.55	...	-3.73
for	-0.99	3.17	...	-3.74
ce	-2.67	-4.23	...	2.07

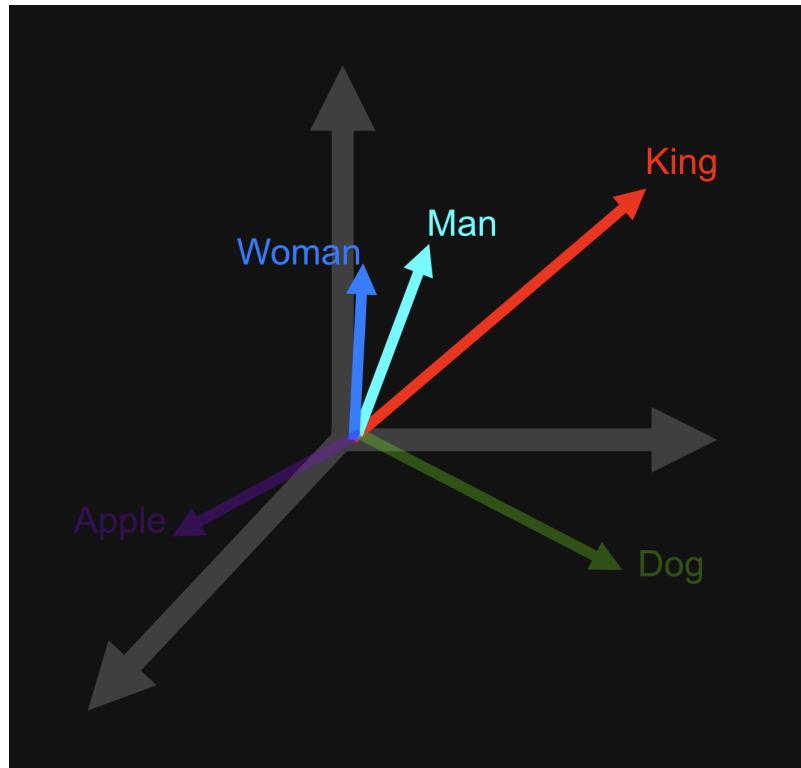
Vectors have meaning encoded into them

- Let's zoom into one of those vectors
- At the beginning, all these numbers are just random
- But when the model trains, some kind of meaning will be encoded in these vectors
- We do not know what exactly each dimension means, the model is like a black box



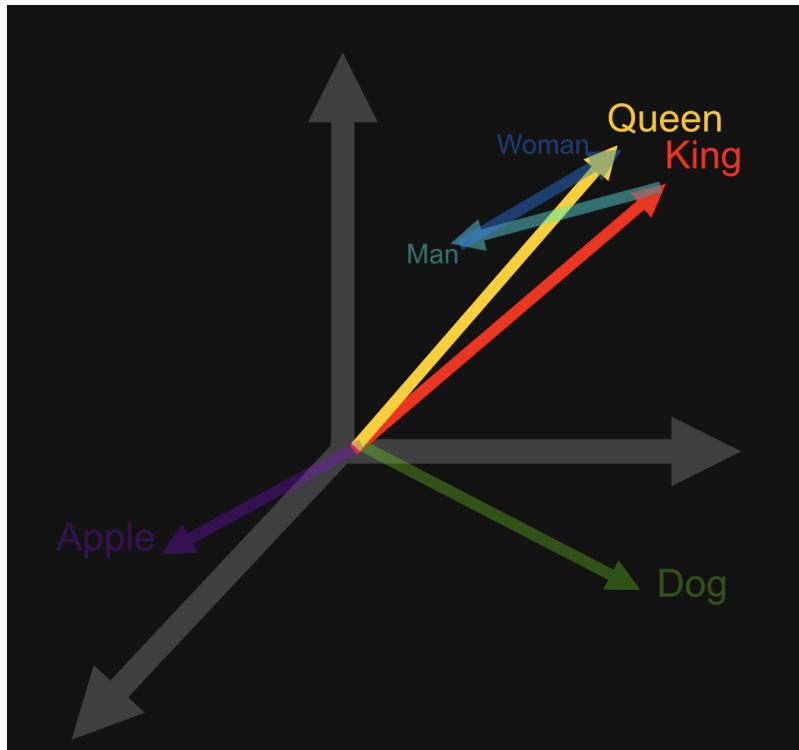
Vectors allow for similarity search

- If the meaning was encoded well, similar words should be near each other in the vector space



Vectors magically understand abstract concepts

- Incredibly, we can do vector math and witness the meaning embedded in the vectors
- King - Man + Woman = Queen

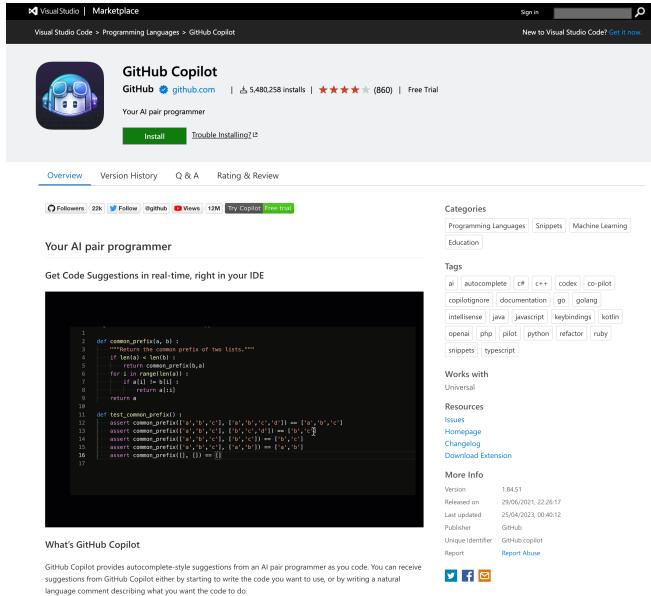
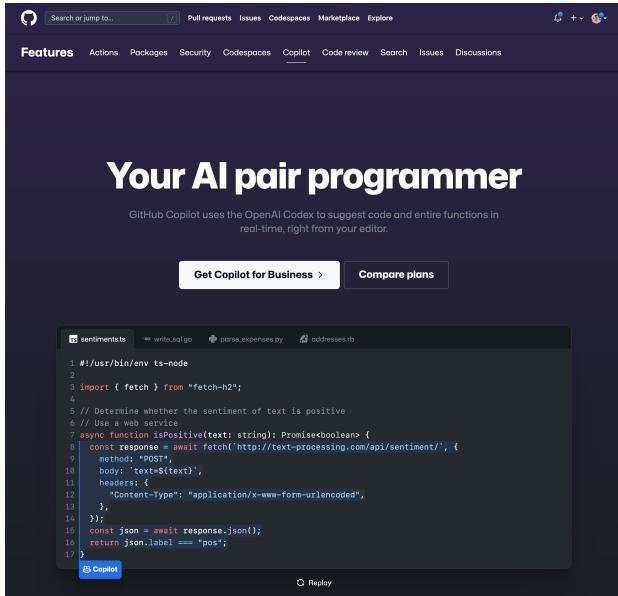


How can I learn more?

- Neural Networks by 3Blue1Brown
- Transformers & NLP by Lucidate
- Visual Guide to Transformer Neural Networks by Hedy AI
- Intuition Behind Self-Attention Mechanism in Transformer Networks by Ark

Installing Github Copilot

- <https://github.com/features/copilot>
- <https://marketplace.visualstudio.com/items?itemName=GitHub.copilot>



Subscribing to OpenAI API

- <https://platform.openai.com/overview>
- <https://platform.openai.com/account/api-keys>
- <https://platform.openai.com/account/usage>

Welcome to the OpenAI platform

Start with the basics

- Quickstart tutorial
- Examples

Build an application

- Chat
- Embeddings
- Image generation
- Speech to text
- Fine-tuning

Build a ChatGPT plugin

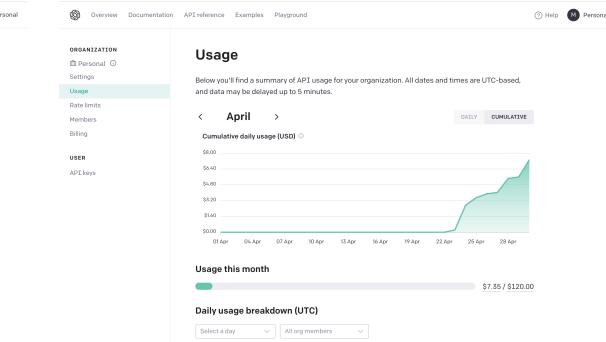
- Introduction
- Examples

API keys

Your secret API keys are listed below. Please note that we do not display your secret API keys again after you generate them.

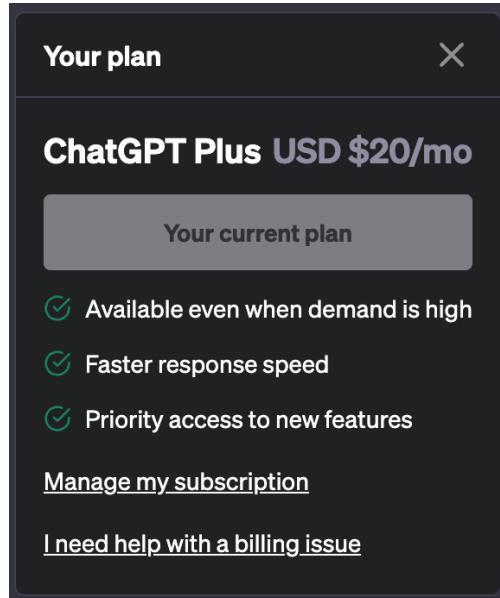
Do not share your API key with others, or expose it in the browser or other client-side code. In order to protect the security of your account, OpenAI may also automatically rotate any API key that we've found has leaked publicly.

NAME	KEY	CREATED	LAST USED
Sacred key	sk-...-4cb0	29 Mar 2023	30 Mar 2023
NLBI-Talk	sk-...-4g7Y	23 Apr 2023	5 May 2023



Subscribing to ChatGPT Plus

- Same login as the OpenAI login: <https://chat.openai.com/auth/login>
- "Profile" --> "My Plan" -> Upgrade to ChatGPT Plus



What we will build today

- We will build a tool that can summarize Youtube videos
- The tool will perform the following steps:
 1. Download the audio of the video
 2. Transcribe the audio file to text using OpenAI's `Whisper` model
 3. Split the large text into many smaller chunks
 4. Summarize each chunk using OpenAI's `text-davinci-003` model

Downloading the audio

- We will use the `yt-dlp` Python library to download the audio
- Our tool will be called via the command line like so:

```
1 python -m gpt_summarize.source_youtube https://www.youtube.com/watch?v=0XcJw5tW9RY
```

- The output will look like this:

```
> python -m gpt_summarize.source_youtube https://www.youtube.com/watch?v=0XcJw5tW9RY
[youtube] Extracting URL: https://www.youtube.com/watch?v=0XcJw5tW9RY
[youtube] 0XcJw5tW9RY: Downloading webpage
[youtube] 0XcJw5tW9RY: Downloading android player API JSON
[info] 0XcJw5tW9RY: Downloading 1 format(s): 251
[dashsegments] Total fragments: 1
[download] Destination: files/audio/Sundar Pichai: AI will have greater impact than electricity | Forum Insight.webm
[download] 100% of 2.72MiB in 00:00:02 at 1.15MiB/s
[ExtractAudio] Destination: files/audio/Sundar Pichai: AI will have greater impact than electricity | Forum Insight.mp3
Deleting original file files/audio/Sundar Pichai: AI will have greater impact than electricity | Forum Insight.webm (pass -k to keep)
Download complete: files/audio/Sundar Pichai: AI will have greater impact than electricity | Forum Insight.mp3
```

Transcribing the audio

- We will use OpenAI's `Whisper` model to transcribe the audio
- We will call another script via the command line:

```
1 python -m gpt_summarize.do_transcribe files/audio/filename.mp3
```

- The output will look like this:

```
> python -m gpt_summarize.do_transcribe "files/audio/Sundar Pichai: AI will have greater impact than electricity | Forum Insight.mp3"
/Users/martin/.pyenv/versions/chatgpt-copilot-whisper/lib/python3.10/site-packages/whisper/transcribe.py:114: UserWarning: FP16 is not supported on CPU; using FP32 instead
[00:00.000 --> 00:04.440] AI is probably the most important thing humanity has ever worked on.
[00:04.440 --> 00:08.920] And I think of it as something more profound than electricity or fire.
[00:08.920 --> 00:14.760] And anytime you work with technology, you need to learn to harness the benefits while
[00:14.760 --> 00:16.440] minimizing the downsides.
[00:16.440 --> 00:21.260] You know, stepping back, when you think about a lot of problems in the world, today it's
[00:21.260 --> 00:24.600] because we typically have a constraint on resources.
[00:24.960 --> 00:30.000] AI for the first time, I think over time, offers a different construct.
[00:30.000 --> 00:36.560] Things which are constrained and look like a zero-sum game today may not be so in the future.
[00:36.560 --> 00:38.240] Take education, for example.
[00:38.240 --> 00:41.880] It's really difficult to educate people in a cost-effective way.
[00:41.880 --> 00:44.920] AI may fundamentally change that equation.
[00:44.920 --> 00:49.840] It may make it possible for us to have clean, cheap, renewable energy for the future.
[00:49.840 --> 00:54.280] So I think a lot of things will play out in more positive ways than people think.
[00:54.280 --> 00:56.320] But the risks are important.
[00:56.320 --> 00:59.520] And I think the way we solve it is we think ahead.
[00:59.520 --> 01:01.160] We worry about it.
[01:01.160 --> 01:07.400] We do things like be upfront, have ethical charters.
[01:07.400 --> 01:09.840] Think about AI safety from day one.
[01:09.840 --> 01:14.360] Be very transparent and open in how we perceive progress there.
```

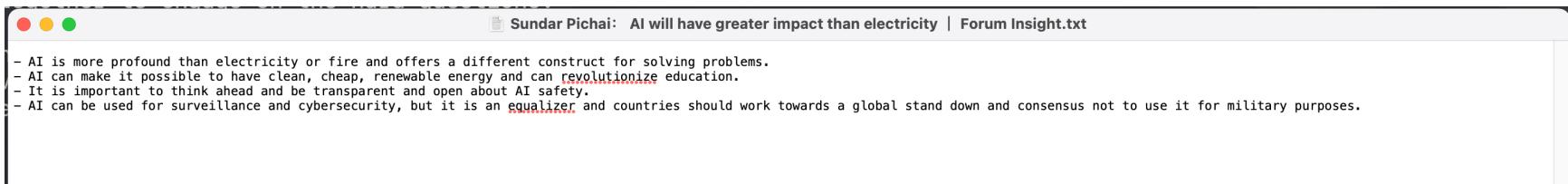
Summarizing the transcript

- We will use OpenAI's `text-davinci-003` model to summarize the transcript
- We will call another script via the command line:

```
1 python -m gpt_summarize.do_summarize files/transcripts/filename.txt
```

- The output will look like this:

```
> python -m gpt_summarize.do_summarize "files/transcripts/Sundar Pichai: AI will have greater impact than electricity | Forum Insight.txt"
Summary saved to files/summaries/Sundar Pichai: AI will have greater impact than electricity | Forum Insight.txt.
```



Creating a project

```
1 # First, let's create a new project folder:  
2 mkdir -p ~/Projects/gpt_summarize/gpt_summarize
```

- NOTE: the `mkdir` command stands for "make directory"

Creating a project

```
1 mkdir -p ~/Projects/gpt_summarize/gpt_summarize  
2  
3 # Let's `cd` into the newly created folder  
4 cd ~/Projects/gpt_summarize
```

- NOTE: the `cd` command stands for "change directory"

Creating a project

```
1 mkdir -p ~/Projects/gpt_summarize/gpt_summarize
2 cd ~/Projects/gpt_summarize
3
4 # Now create a few files that we will need later:
5 touch .gitignore
6 touch gpt_summarize/__init__.py
7 touch gpt_summarize/source_youtube.py
8 touch gpt_summarize/do_transcribe.py
9 touch gpt_summarize/do_summarize.py
10 touch gpt_summarize/local_settings.py
```

- NOTE: the `touch` command creates an empty file

Creating a project

```
1  mkdir -p ~/Projects/gpt_summarize/gpt_summarize
2  cd ~/Projects/gpt_summarize
3  touch .gitignore
4  touch gpt_summarize/__init__.py
5  touch gpt_summarize/source_youtube.py
6  touch gpt_summarize/do_transcribe.py
7  touch gpt_summarize/do_summarize.py
8  touch gpt_summarize/local_settings.py
9
10 # We will also need a few folders for all the files that our tool will generate:
11 mkdir -p files/audio
12 mkdir -p files/transcripts
13 mkdir -p files/summaries
14 `
```

Creating a project

- When you run `tree . -a`, your file structure should look like this:

```
1 .
2   ├── .gitignore
3   └── files
4     ├── audio
5     ├── summaries
6     └── transcripts
7     └── gpt_summarize
8       ├── __init__.py
9       ├── do_summarize.py
10      ├── do_transcribe.py
11      ├── local_settings.py
12      └── source_youtube.py
13
14 5 directories, 6 files
```

- NOTE: the `tree .` command shows the file structure of the current folder
 - the `-a` flag shows hidden files like the `.`gitignore` file

The ` `.gitignore` file

- Put the following code into the ` `.gitignore` file:

```
1 local_settings.py  
2 files/  
3 __pycache__/  
4 .DS_Store
```

The ` `.gitignore` file

- Put the following code into the ` `.gitignore` file: model on it.

```
1 local_settings.py  
2 files/  
3 __pycache__/  
4 .DS_Store
```

- The most important line is the first one, it makes sure that Github Copilot does never look into the `local_settings.py` file
- IMPORTANT: When working with Copilot you must be aware that VSCode will constantly send your code to Github's servers
 - Make sure that your code does not contain sensitive information
 - Make sure that your company allows you to use Copilot
 - If your company hosts code on Github anyways, there is no reason not to use Copilot

The `local_settings.py` file

- Put the following code into the `local_settings.py` file:

```
1 OPENAI_API_KEY = 'YOUR KEY HERE'
```

- You can get your API key [here](#)
- This is considered secret information, which is why we have this file in ` .gitignore`
- Never share this key with anyone or they can use your OpenAI credits

The `__main__` block for `source_youtube.py`

- Put the following code into the `source_youtube.py` file:

```
1 import sys
2
3 if __name__ == "__main__":
4     if len(sys.argv) > 1:
5         url = sys.argv[1]
6         audio_path = download_audio(url)
7         print(f"Download complete: {audio_path}")
8     else:
9         print("Usage: python -m gpt_summarize.source_youtube <youtube_url>")
```

The `__main__` block for `source_youtube.py`

- Put the following code into the `source_youtube.py` file:

```
1 import sys
2
3 if __name__ == "__main__":
4     if len(sys.argv) > 1:
5         url = sys.argv[1]
6         audio_path = download_audio(url)
7         print(f"Download complete: {audio_path}")
8     else:
9         print("Usage: python -m gpt_summarize.source_youtube <youtube_url>")
```

- The variable `__name__` is a special variable in Python
- It will have the string value ` "__main__ "` if this file is executed directly
 - If we call this script from the command line, the value will be ` "__main__ "`
 - If we import any function or variable from this file into another file, the value will be ` "gpt_summarize.source_youtube" `

The `__main__` block for `source_youtube.py`

- Put the following code into the `source_youtube.py` file:

```
1 import sys
2
3 if __name__ == "__main__":
4     if len(sys.argv) > 1:
5         url = sys.argv[1]
6         audio_path = download_audio(url)
7         print(f"Download complete: {audio_path}")
8     else:
9         print("Usage: python -m gpt_summarize.source_youtube <youtube_url>")
```

- NOTE: The code will not work, because we have not yet implemented the function `download_audio`

Adding some dummy functions

- Let's add a dummy function to `source_youtube.py`:

```
1 import sys, os
2
3 def download_audio(url):
4     return "files/audio/test.webm"
5
6 if __name__ == "__main__":
7     if len(sys.argv) > 1:
8         url = sys.argv[1]
9         audio_path = download_audio(url)
10        print(f"Download complete: {audio_path}")
11    else:
12        print("Usage: python -m gpt_summarize.source_youtube <youtube_url>")
```

- Now we can run the code and see if it works:

```
1 python -m gpt_summarize.source_youtube test`
```

CY

How can I download the audio of a Youtube with yt-dlp in Python?



2. Create a Python script and import the required libraries:

```
python  
  
import yt_dlp  
import os
```

 Copy code

3. Define a function to download the audio from a YouTube video:

```
python  
  
def download_audio(url):  
    # Configure yt-dlp options  
    ydl_opts = {  
        'format': 'bestaudio/best',  
        'postprocessors': [{  
            'key': 'FFmpegExtractAudio',  
            'preferredcodec': 'mp3',  
            'preferredquality': '192',  
        }],  
        'outtmpl': 'downloads/%(title)s.%s.(ext)s',  
        'quiet': True,  
    }  
  
    # Download the audio  
    with yt_dlp.YoutubeDL(ydl_opts) as ydl:  
        ydl.download([url])
```

 Copy code

The `download_audio` function

- Replace the `download_audio` function with the following code:
- Also add the import statement for `yt_dlp` at the top of the file

```
1  import sys
2
3  import yt_dlp
4
5  def download_audio(url):
6      ydl_opts = {
7          "format": "bestaudio/best",
8          "outtmpl": "files/audio/%(title)s.%s%(ext)s",
9          "postprocessors": [
10              {
11                  "key": "FFmpegExtractAudio",
12                  "preferredcodec": "mp3",
13                  "preferredquality": "192",
14              }
15          ],
16      }
17
18      with yt_dlp.YoutubeDL(ydl_opts) as ydl:
19          ydl.download([url])
```

CY

How can I retrieve the filename that yt-dlp generated?





To retrieve the filename that yt-dlp generated, you can use the `progress_hooks` option.



You can create a custom function that will be called by yt-dlp during the download process, allowing you to store the final filename.

Here's an example of how to do this:

```
python

import yt_dlp
import os

def download_hook(d):
    if d['status'] == 'finished':
        print(f"Downloaded: {d['filename']}")

def download_audio(url):
    ydl_opts = {
        'format': 'bestaudio/best',
        'postprocessors': [{
            'key': 'FFmpegExtractAudio',
            'preferredcodec': 'mp3',
            'preferredquality': '192',
        }],
        'outtmpl': 'downloads/%(title)s.%(ext)s',
        'quiet': True,
        'progress_hooks': [download_hook],
    }

    with yt_dlp.YoutubeDL(ydl_opts) as ydl:
        ydl.download([url])
```

The `download_audio` function

- Replace the `download_audio` function with the following code:
- Also add the import statement for `os` at the top of the file

```
1  import sys
2  import os
3
4  import yt_dlp
5
6  def download_audio(url):
7      output_filename_result = {}
8
9      def get_filename(info_dict):
10          if info_dict["status"] != "finished":
11              return None
12
13          filename = info_dict["filename"]
14          filename, ext = os.path.splitext(filename)
15          output_filename_result["file_path"] = filename + ".mp3"
16
17      ydl_opts = {
18          "format": "bestaudio/best",
19          "outtmpl": "files/audio/%(title)s.%(ext)s",
```

Try it out!

- Now we can run the code and see if it works:

```
1 python -m gpt_summarize.source_youtube https://www.youtube.com/watch?v=0XcJw5tW9RY
```

The `__main__` block for `do_transcribe.py`

- Like before, we can add a `__main__` block to `do_transcribe.py`:
- This time, we will not provide a Youtube URL, but an audio file path as a command line argument

```
1 import os, sys, whisper
2
3 if __name__ == "__main__":
4     if len(sys.argv) > 1:
5         audio_path = sys.argv[1]
6         text_path = transcribe(audio_path)
7         print(f"Transcript saved to: {text_path}.")
8     else:
9         print("Usage: python -m gpt_summarize.do_transcribe <audio_path>")
```

The `__main__` block for `do_transcribe.py`

- Like before, we can add a `__main__` block to `do_transcribe.py`:
- This time, we will not provide a Youtube URL, but an audio file path as a command line argument

```
1 import os, sys, whisper
2
3 if __name__ == "__main__":
4     if len(sys.argv) > 1:
5         audio_path = sys.argv[1]
6         text_path = transcribe(audio_path)
7         print(f"Transcript saved to: {text_path}.")
8     else:
9         print("Usage: python -m gpt_summarize.do_transcribe <audio_path>")
```

- NOTE: Once again, this code will crash because we have not implemented the `transcribe` function yet

The `__main__` block for `do_transcribe.py`

- Let's add a dummy function to `do_transcribe.py`:

```
1 def transcribe(audio_path):
2     return "files/transcripts/test.txt"
3
4
5 if __name__ == "__main__":
6     if len(sys.argv) > 1:
7         audio_path = sys.argv[1]
8         text_path = transcribe(audio_path)
9         print(f"Transcript saved to: {text_path}.")
10    else:
11        print("Usage: python -m gpt_summarize.do_transcribe <audio_path>")
```

- Now we can run the code and see if it works:

```
1 python -m gpt_summarize.do_transcribe files/audio/test.mp3
```

The `transcribe` function

- Check <https://github.com/openai/whisper> to learn how to use Whisper
- Replace the `transcribe` function with the following code:

```
1 def transcribe(audio_path):
2     model = whisper.load_model("small.en")
3     result = model.transcribe(audio_path, verbose=True)
4     transcript = str(result["text"])
5
6     filename = os.path.splitext(os.path.basename(audio_path))[0]
7     text_path = os.path.join("files/transcripts", f"{filename}.txt")
8     with open(text_path, "w") as f:
9         f.write(transcript)
10
11     return text_path
```

- Run the code!

```
1 python -m gpt_summarize.do_transcribe "files/audio/Sundar Pichai: AI will have greater impact than electricity |
```

CY

Please explain this line of code:



```
filename = os.path.splitext(os.path.basename(audio_path))[0]
```



This line of code is written in Python and uses the `os` library, specifically the `os.path` module, which provides functionality for working with file paths. The line of code processes the input `audio_path` to extract the filename without its extension.



Here's a breakdown of the code:

1. `os.path.basename(audio_path)`: This function extracts the base name (i.e., the file name without the directory path) of the input `audio_path`. For example, if `audio_path` is `/home/user/audio/sample.wav`, the result will be `sample.wav`.
2. `os.path.splitext(...)`: This function splits the input file name into a tuple containing two parts: the file name without the extension and the extension itself. In the example above, the input is `sample.wav`, so the result will be a tuple: `('sample', '.wav')`.
3. `[0]`: This indexing operation retrieves the first element of the tuple, which is the file name without the extension. In the example above, it retrieves the string `''sample'`.

So, the line of code effectively extracts the base file name without the extension from the given `audio_path` and stores it in the variable `filename`.

The `__main__` block for `do_summarize.py`

- Just like before, we can add a `__main__` block to `do_summarize.py`

```
1 import os, sys, openai, spacy, tiktoken
2
3 if __name__ == "__main__":
4     if len(sys.argv) > 1:
5         text_path = sys.argv[1]
6         sentences = get_sentences(text_path)
7         chunks = get_chunks(sentences)
8         filename = os.path.splitext(os.path.basename(text_path))[0]
9         summary_path = summarize(chunks, filename)
10
11         print(f"Summary saved to {summary_path}.")
12     else:
13         print(
14             "Usage: python -m gpt_summarize.do_summarize <transcript_path>"
15         )
```

The `__main__` block for `do_summarize.py`

- Just like before, we can add a `__main__` block to `do_summarize.py`

```
1 import os, sys, openai, spacy, tiktoken
2
3 if __name__ == "__main__":
4     if len(sys.argv) > 1:
5         text_path = sys.argv[1]
6         sentences = get_sentences(text_path)
7         chunks = get_chunks(sentences)
8         filename = os.path.splitext(os.path.basename(text_path))[0]
9         summary_path = summarize(chunks, filename)
10
11         print(f"Summary saved to {summary_path}.")
12     else:
13         print(
14             "Usage: python -m gpt_summarize.do_summarize <transcript_path>"
15         )
```

- NOTE: As usual, this code will crash because we have not implemented the `get_sentences` and `get_chunks` and `summarize` functions yet

The `__main__` block for `do_summarize.py`

- Let's add dummy functions to `do_summarize.py`:

```
1 # imports here...
2
3 def get_sentences(text_path):
4     return ["This is a sentence."]
5
6 def get_chunks(sentences):
7     return ["This is a chunk."]
8
9 def summarize(chunks, filename):
10    return 'files/summaries/test.txt'
11
12 # __main__ block here...
```

- Run the code and see if it works:

```
1 python -m gpt_summarize.do_summarize files/transcripts/test.txt
```

Some constants

- At the top of `do_summarize.py`, after the imports, add the following code:
- See <https://platform.openai.com/docs/models>
- See <https://openai.com/pricing>

```
1 # other imports...
2
3 from .local_settings import OPENAI_API_KEY
4
5 PROMPT = """Create a bullet point summary of the following text.
6 Make sure that all major talking points are part of the summary.
7 Use '- ' for bullet points:
8
9 {chunk}
10 """
11
12 MODEL = "text-davinci-003"
13 ENCODING = "cl100k_base"
14 MODEL_MAX_TOKENS = 4096
15 COST_PER_1K_TOKENS_USD = 0.02
16 RESPONSE_TOKENS = 1000
17
18 # rest of the code...
```

Working with tokens

- See [What are tokens and how to count them](#)

What are tokens?

Tokens can be thought of as pieces of words. Before the API processes the prompts, the input is broken down into tokens. These tokens are not cut up exactly where the words start or end - tokens can include trailing spaces and even sub-words. Here are some helpful rules of thumb for understanding tokens in terms of lengths:

- 1 token \approx 4 chars in English
- 1 token \approx $\frac{3}{4}$ words
- 100 tokens \approx 75 words

Or

- 1-2 sentence \approx 30 tokens
- 1 paragraph \approx 100 tokens
- 1,500 words \approx 2048 tokens

Working with tokens

- See [Tiktoken](#)

The screenshot shows the `README.md` file for the `tiktoken` library. It features a header with a sand timer icon and the word `tiktoken`. Below the header, a brief description states: "tiktoken is a fast BPE tokeniser for use with OpenAI's models." A code block demonstrates how to import the library and use its `get_encoding` method:

```
import tiktoken
enc = tiktoken.get_encoding("cl100k_base")
assert enc.decode(enc.encode("hello world")) == "hello world"
```

Below this, another code block shows how to get a specific model's tokeniser:

```
# To get the tokeniser corresponding to a specific model in the OpenAI API:
enc = tiktoken.encoding_for_model("gpt-4")
```

Further down, it says: "The open source version of `tiktoken` can be installed from PyPI:" followed by the command:

```
pip install tiktoken
```

At the bottom, it notes: "The tokeniser API is documented in `tiktoken/core.py` ."

Working with tokens

- See <https://platform.openai.com/docs/models/gpt-3-5>
- The `text-davinci-003` model has a maximum token length of 4,097
- When we send a prompt to the API, we must make sure that the PROMPT + RESPONSE fits into the maximum token limit
 - Example: If our prompt is 4,000 tokens, then ChatGPT can only respond with 97 tokens

Working with tokens

- Our prompt will look like this
- If the `chunk` variable is empty, this would equal 37 tokens:

```
1 PROMPT = """Create a bullet point summary of the following text.  
2 Make sure that all major talking points are part of the summary.  
3 Use '- ' for bullet points:  
4  
5 {chunk}  
6 """
```

```
1 Prompt: 37 tokens
```

Working with tokens

```
1 PROMPT = """Create a bullet point summary of the following text.  
2 Make sure that all major talking points are part of the summary.  
3 Use '- ' for bullet points:  
4  
5 {chunk}  
6 """
```

- We want to leave room for 1000 tokens for ChatGPT to respond

```
1 Prompt:    37 tokens  
2 Response: 1000 tokens
```

Working with tokens

```
1 PROMPT = """Create a bullet point summary of the following text.  
2 Make sure that all major talking points are part of the summary.  
3 Use '- ' for bullet points:  
4  
5 {chunk}  
6 """
```

- That means our chunks can have a maximum size of ``4097 - 1037 = 3060`` tokens

```
1 Prompt:    37 tokens  
2 Chunk:     3060 tokens  
3 Response:  1000 tokens  
4 -----  
5 Total:     4097 tokens
```

CY

How can I split a large text into its sentences using spaCy?

- *NOTE: I happen to know that spaCy is a good way to split text into sentences*



To split a large text into its sentences using spaCy, you'll first need to install the library and a language model. In this example, we'll use the English language model. Follow these steps:

1. Install spaCy and the English language model:

Copy code

```
pip install spacy
python -m spacy download en_core_web_sm
```

2. Create a Python script or use an interactive Python environment, such as Jupyter Notebook, to write and run the following code:

Copy code

```
python

import spacy

# Load the English language model
nlp = spacy.load("en_core_web_sm")

# Define the text you want to split into sentences
text = "This is an example text. It contains multiple sentences. You can rep

# Process the text using the loaded language model
doc = nlp(text)

# Iterate over the sentences and print them
for sent in doc.sents:
    print(sent.text)
```

The `get_sentences` function

- Replace the `get_sentences` function with the following code:

```
1 def get_sentences(text_path):  
2     with open(text_path, "r") as f:  
3         text = f.read()  
4     nlp = spacy.load("en_core_web_sm")  
5     spacy_docs = nlp(text)  
6     sentences = [sentence.text.strip() for sentence in spacy_docs.sents]  
7     return sentences
```

The `get_sentences` function

- Replace the `get_sentences` function with the following code:

```
1 def get_sentences(text_path):  
2     with open(text_path, "r") as f:  
3         text = f.read()  
4     nlp = spacy.load("en_core_web_sm")  
5     spacy_docs = nlp(text)  
6     sentences = [sentence.text.strip() for sentence in spacy_docs.sents]  
7     return sentences
```

- If you are new to Python, this line will probably be confusing!

CY

Please explain this line of code:



```
sentences = [sentence.text.strip() for sentence in spacy_docs.sents]
```



This line of code is a list comprehension that creates a list of sentences from a given spaCy `Doc` object (`spacy_docs`). Let's break it down step by step:

1. `spacy_docs.sents`: This is an iterator over the sentences in the `spacy_docs` object, which is a `Doc` object processed by spaCy. It contains the sentences identified in the text after tokenization and sentence segmentation.
2. `sentence.text`: This code iterates over each `sentence` in the `spacy_docs.sents` iterator. The `text` attribute of each `sentence` (which is a `Span` object) is used to extract the raw text of the sentence.
3. `sentence.text.strip()`: The `strip()` method is called on the raw text of each sentence. This method removes any leading and trailing whitespaces (including spaces, tabs, and newlines) from the sentence text.
4. ` [...] for sentence in spacy_docs.sents]`: This part of the code is the list comprehension. It iterates over each `sentence` in `spacy_docs.sents`, applies the `sentence.text.strip()` operation, and appends the result to a new list.

The result is a list of sentences as strings, with leading and trailing whitespaces removed. This list is assigned to the variable `sentences`.



CY

Thank's a lot, you smart cookie! <3



You're welcome! I'm glad I could help. If you have any more questions or need further assistance, feel free to ask. Have a great day! 😊



The `count_tokens` function

- Before we can implement the `get_chunks` function, we need a way to count the number of tokens in a chunk
- See [Tiktoken README](#)
- Add this `count_tokens` function above all other functions:

```
1 def count_tokens(text):
2     enc = tiktoken.encoding_for_model(MODEL)
3     tokens = enc.encode(text)
4     token_count = len(tokens)
5     return token_count
```

CY

I have this code:



```
...
PROMPT = """
Create a bullet point summary of the following text.
Make sure that all major talking points are part of the summary.
Use '- ' for bullet points:
```

```
{chunk}
"""
MODEL_MAX_TOKENS = 4096
RESPONSE_TOKENS = 1000
```

```
def count_tokens(text):
    enc = tiktoken.encoding_for_model(MODEL)
    tokens = enc.encode(text)
    token_count = len(tokens)
    return token_count
...
```

Please implement a `get_chunks` function that takes an array of strings where each string is an English sentence. It should return an array of chunks.

I will use this array of chunks to send API calls to OpenAI and the token limit for those calls is `MODEL_MAX_TOKENS`. When generating the chunks, you need to make sure that each chunk is small enough so that I can insert it into the prompt and still have 1000 tokens left for the response.



You can implement the `get_chunks` function as follows:



```
python
```

```
def get_chunks(sentences):
    chunks = []
    current_chunk = ""

    for sentence in sentences:
        # Check if adding the sentence to the current chunk would exceed the
        new_chunk = f"{current_chunk} {sentence}"
        new_chunk_token_count = count_tokens(PROMPT.format(chunk=new_chunk))

        if new_chunk_token_count <= MODEL_MAX_TOKENS:
            # If not, add the sentence to the current chunk
            current_chunk = new_chunk.strip()
        else:
            # If yes, start a new chunk with the sentence
            chunks.append(current_chunk)
            current_chunk = sentence.strip()

    # Add the last chunk if it's not empty
    if current_chunk:
        chunks.append(current_chunk)

    return chunks
```

Copy code

The `get_chunks` function

- Replace the `get_chunks` function with the following code:

```
1  def get_chunks(sentences):
2      chunks = []
3      current_chunk = ""
4
5      for sentence in sentences:
6          # Check if adding the sentence to the current chunk would exceed the token limit
7          new_chunk = f"{current_chunk} {sentence}"
8          new_chunk_token_count = (
9              count_tokens(PROMPT.format(chunk=new_chunk)) + RESPONSE_TOKENS
10         )
11
12         if new_chunk_token_count <= MODEL_MAX_TOKENS:
13             # If not, add the sentence to the current chunk
14             current_chunk = new_chunk.strip()
15         else:
16             # If yes, start a new chunk with the sentence
17             chunks.append(current_chunk)
18             current_chunk = sentence.strip()
19
20         # Add the last chunk if it's not empty
21         if current_chunk:
22             chunks.append(current_chunk)
```

The `summarize` function

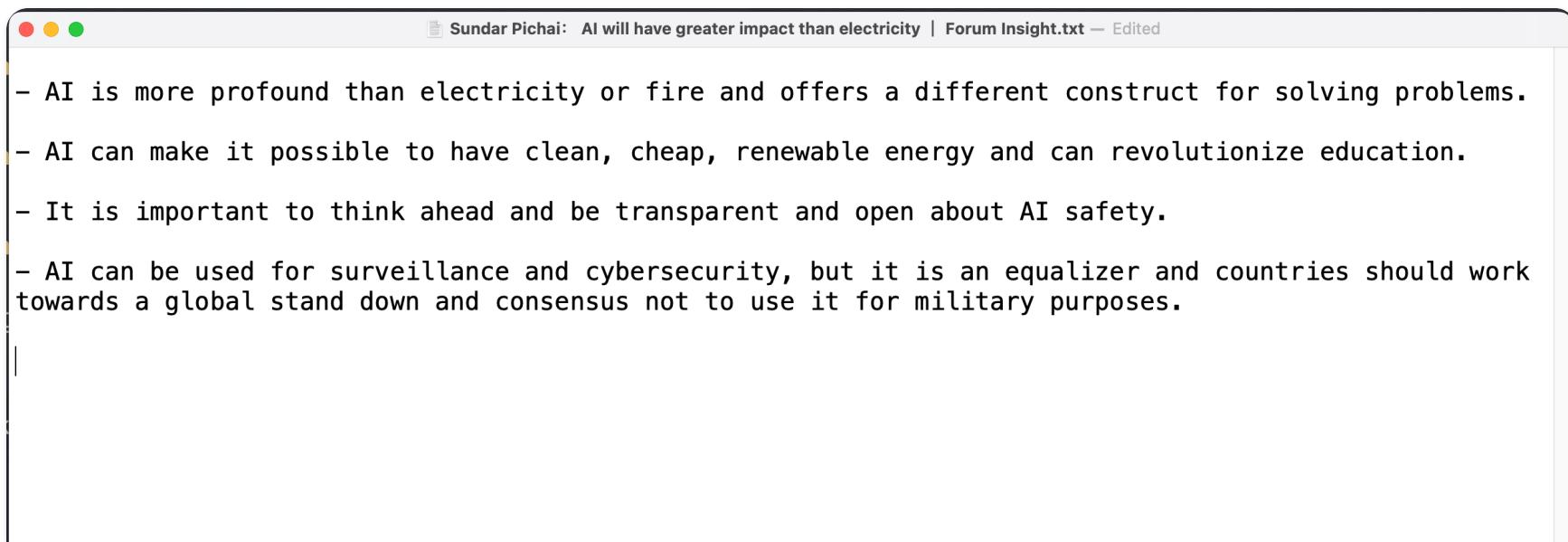
- See OpenAI API docs about completion
- Replace the `summarize` function with the following code:

```
1  def summarize(chunks, filename):
2      openai.api_key = OPENAI_API_KEY
3      summaries = []
4      for chunk in chunks:
5          prompt = PROMPT.format(chunk=chunk)
6          prompt_tokens = count_tokens(prompt)
7          result = openai.Completion.create(
8              model="text-davinci-003",
9              prompt=prompt,
10             # max_tokens refers to the max tokens of the RESPONSE!!!
11             max_tokens=MODEL_MAX_TOKENS - prompt_tokens,
12             temperature=0,
13             n=1,
14         )
15         summaries.append(result)
16
```

Demo

- Our program should be fully functional now!
- Let's give it a try:

```
1 python -m gpt_summarize.source_youtube https://www.youtube.com/watch?v=0XcJw5tW9RY
2 python -m gpt_summarize.do_transcribe "files/audio/Sundar Pichai: AI will have greater impact than electricity |"
3 python -m gpt_summarize.do_summarize "files/transcripts/Sundar Pichai: AI will have greater impact than electrici
```



```
Sundar Pichai: AI will have greater impact than electricity | Forum Insight.txt — Edited

- AI is more profound than electricity or fire and offers a different construct for solving problems.
- AI can make it possible to have clean, cheap, renewable energy and can revolutionize education.
- It is important to think ahead and be transparent and open about AI safety.
- AI can be used for surveillance and cybersecurity, but it is an equalizer and countries should work towards a global stand down and consensus not to use it for military purposes.
```

Thank you for your attention!

- Join the Python User Group: <https://pugs.org.sg/membership>
- Follow me on Twitter: <https://twitter.com/@mbrochh>

