Masterclass in
OPEN AI, GPT, and DALL.E

Practical Coding and Usage Course

What is OPEN AI?

Introduction

- Text Completion or Generation (GPT)
- Image Creation (DALL.E)
- Code Completion (Codex)
- ChatGPT

- Moderation (free to use)

Getting Started (Code Snippet)

```
import openai

openai.Completion.create(
  engine="davinci",
  prompt="Make a list of horror movies:"
)
```

Text Completion Model

- What it can do

- Ada
- Babbage
- Curie
- Davinci

- Cost based on the model
- Speed and latency based on model
- Price per 1,000 tokens

Codex Model

- Python, Javascript, Go, Perl, PHP, Ruby, Swift, Typescript, SQL, Shell

- Davinci
- Cushman (Smaller version of Davinci)

Tokens

- What are Tokens?
  - Method of billing – based on engine
  - Units of 1,000

  - Typically 0.0004 to 0.02 cents per 1k tokens
  - More expensive for fine-tuned and embedded models

  - Free credits to get started and then pay as you go
  - Quota to start
  - Billed for request and response tokens

DALL.E Image Model

- What it can do

- Currently 256x256, 512x512, 1024x1024
- Cost based on size
- Typically 1 or 2 cents

Some Conventions (I Use)

- GPT / GPT-3 / OPENAI / AI

- One Shot
- Few Shot

- Library / HTTP Request / API / JSON

- Request / Question / Prompt / Pattern / Instruction

Playground

- Uses credits – but doesn't require code
- Be careful of "best_of" and "n" requests
- Check the website for latest pricing, quotas, and limitations

Signing Up

- Free Credits (No Credit Card)
- Country Limitations
- Phone Verification

- Use the Playground to get familiar

Prompt Engineering

Ask a Question or define a pattern

A Simple Question

- Ask a question and get a reply

What colour are fire engines in the UK?

In the UK, fire engines are usually red.

- Responses vary

Generation Some Ideas

Brainstorm some ideas combining VR and fitness:

Simple completion

Vertical farming provides a novel solution for producing food locally, reducing transportation costs and

Classification

Decide whether a Tweet's sentiment is positive, neutral, or negative.

Tweet: I loved the new Batman movie!

Sentiment:

Multiple Requests

Classify the sentiment in these tweets:

1. "I can't stand homework"
2. "This sucks. I'm bored 😬"
3. "I can't wait for Halloween!!!"
4. "My cat is adorable ❤❤"

Tweet sentiment ratings:

1.

Example : Grammar correction

Correct this to standard English:

She no went to the market.

She did not go to the market.

Summarization

Summarize this for a second-grade student:

Jupiter is the fifth planet from the Sun and the largest in the Solar System. It is a gas giant with a mass one-thousandth that of the Sun, but two-and-a-half times that of all the other planets in the Solar System combined. Jupiter is one of the brightest objects visible to the naked eye in the night sky, and has been known to ancient civilizations since before recorded history. It is named after the Roman god Jupiter.[19] When viewed from Earth, Jupiter can be bright enough for its reflected light to cast visible shadows,[20] and is on average the third-brightest natural object in the night sky after the Moon and Venus.

Convert Movie into emoji

Convert movie titles into emoji.

Back to the Future: 👨👴🚗🕒

Batman: 🤵 🦇

Transformers: 🚗🤖

Star Wars:

Example : Language Translation

Translate this into 1. French, 2. Spanish and 3. Japanese: What rooms do you have available?

1. Quels sont les chambres que vous avez disponibles?
2. ¿Qué habitaciones tienes disponibles?
3. どの部屋が利用可能ですか？

Example : SQL translate

```
### Postgres SQL tables, with their properties:
# # Employee(id, name, department_id)
# Department(id, name, address)
# Salary_Payments(id, employee_id, amount, date)
#
### A query to list the names of the departments which employed more than 10 employees in the last 3 months

SELECT DISTINCT department.name
FROM department
JOIN employee ON department.id = employee.department_id
JOIN salary_payments ON employee.id = salary_payments.employee_id
WHERE salary_payments.date > (CURRENT_DATE - INTERVAL '3 months')
GROUP BY department.name
HAVING COUNT(employee.id) > 10;
```

Improving the Response

- Be explicit
  - Show and tell
  - Give more example responses
  - Provide quality data
  - Use “temperature” and “top_p” settings

Be Very Specific

- About desired content, outcome, length, format, style etc

Write a poem about OpenAI

vs

Write a short inspiring poem about Open AI, focusing on the recent DALL-E product launch (DALL-E is a text to image ML model) in the style of Shakespeare

Give Examples (Pt 1)

Extract the entities mentioned in the text below. Extract the following 4 entity types: company names, people names, specific topics and themes.

Text: {text}

Give Examples (Pt 2)

Extract the important entities mentioned in the text below. First extract all company names, then extract all people names, then extract specific topics which fit the content and finally extract general overarching themes

Desired format:

Company names: -,-

People names: -,-

Specific topics: -,-

General themes: -,-

Text: {text}

Reduce Fluff (Be precise)

The description for this product should be fairly short, a few sentences only, and not too much more.

Vs

Use a 3 to 5 sentence paragraph to describe this product.

Positive Instruction

- Tell the AI what to do instead of what NOT to do

The following is a conversation between an Agent and a Customer. DO NOT ASK USERNAME OR PASSWORD. DO NOT REPEAT.

Customer: I can’t log in to my account.

Agent:

vs

The following is a conversation between an Agent and a Customer. The agent will attempt to diagnose the problem and suggest a solution, whilst refraining from asking any questions related to PII. Instead of asking for PII, such as username or password, refer the user to the help article www.samplewebsite.com/help/faq

Customer: I can’t log in to my account.

Agent:

Lead the Way

```
# Write a simple python function that

# 1. Ask me for a number in mile

# 2. It converts miles to kilometers
```

Vs

```
# Write a simple python function that
# 1. Ask me for a number in mile
# 2. It converts miles to kilometers

import
```

Maybe Not Math Yet

- solve for x where $3x + 4 = 66$

Given the algebraic equation below, solve for the provided variable. Provide the answer in the format of ‘x=<insert answer>.

Problem1: y-1=0, solve for y

Answer1: y=1

Problem2: 3x+4=66, solve for x.

Answer2: x=

- wolframalpha.com

Number of Responses

Science Fiction books:

1. Dune by Frank Herbert
2.

vs

A list of 5 Science Fiction books:

1. Dune by Frank Herbert
2.

Adjusting Tone

- Response:
- Polite Response:

- Including more example answers in the request
  - Note : Uses tokens

- “In the style of …” or “written by …”

Stopping Drift

- Make sure you use “temperature” or “top_p”
- Especially if you ask lots of questions
- Helps the AI stay on topic

Position of the \n matters

Human: Hello, who are you?\n
AI: I am an AI created by OpenAI. How can I help you today?\n
Human: Can I ask you

vs

Human: Hello, who are you?\n
AI: I am an AI created by OpenAI. How can I help you today?\n
Human: Can I ask you\n

and

ideas vs IDEAS

Smaller Tasks

- Write an article about …

Vs

- Create an outline for an article about …
- Write an introduction paragraph about …
- Write a paragraph about <caption1>
  - Consider breaking this down into even smaller tasks
- Write a conclusion for an article about … with a call to action of …

Getting Longer Text

- longer response
- add more description
- in more details
- elaborate it
- explain it
- provide detailed explanation
- explain the case in more detail
- give more information
- give more details
- provide an additional example
- in more detail
- explain more
- provide more information
- explain your reasoning
- explain why
- explain your answer

Text Completion

- Make the intent clear
- Add some examples so the AI can recognize a pattern or determine valid options
- Start some text or describe a problem (with or without some rules)
- Leave something for the AI to complete

Prompt Design 1 – Templating

1. Your are an assistant helping
2. to generate a blog post
3. from a prompt.

4. Use this format …..
5. Output definition
6. The prompt itself – followed by new line

Who or what are you

"""

You are an assistant helping …

What are you doing

… to generate a blog post …

Output format

… based on a prompt

Use this format, replacing text in brackets with the result. Do not include the brackets in the output:

Instructions with Brackets

Blog post:

# [Title of blog post]

[Content based on the prompt]

"""

Put it all together

```
"""
You are an assistant helping to generate a blog post based on a prompt.
Use this format, replacing text in brackets with the result. Do not include the brackets in the output:

Blog post:
# [Title of blog post]
[Content based on the prompt]
"""
Blue Snowball\n
```

to draft a blog post.

The blog post should have an introduction, 3 topic paragraphs, and a conclusion.

Blog post:
# [Topic of the blog post]
[Blog post body]

to draft a blog post

""""

You are an assistant helping to draft a blog post based on a prompt. The blog post should have an introduction, 3 topic paragraphs, and a conclusion.

Use this format, replacing text in brackets with the result. Do not include the brackets in the output:

Blog post:
# [Title of blog post]
[Content based on the prompt]
""""

Blue Snowball

Recipe parts

"""

You are an assistant helping to AAAA based on a prompt. BBBB

Use this format, replacing text in brackets with the result. Do not include the brackets in the output:

CCCC:

DDDD

"""

<prompt always goes here>

to summarize a document

Summary:

[One paragraph summary of the document]

Recipe parts

"""

You are an assistant helping to summarize a document based on a prompt.

Use this format, replacing text in brackets with the result. Do not include the brackets in the output:

CCCC:

DDDD

"""

<prompt always goes here>

Recipe parts

"""

You are an assistant helping to summarize a document based on a prompt.

Use this format, replacing text in brackets with the result. Do not include the brackets in the output:

Summary:

DDDD

"""

<prompt always goes here>

Recipe parts

"""

You are an assistant helping to summarize a document based on a prompt.

Use this format, replacing text in brackets with the result. Do not include the brackets in the output:

Summary:

[One paragraph summary of the document]

"""

<prompt always goes here>

to write an essay

Essay:

[Introduction paragraph]

## [Name of topic 1]

[Paragraph about topic 1]

## [Name of topic 2]

[Paragraph about topic 2]

## [Name of topic 3]

[Paragraph about topic 3]

## ['Conclusion']

[Conclusion paragraph]

to brainstorm a list of ideas

5 ideas based on the prompt:

- [Idea 1]
- [Idea 2]
- [Idea 3]
- [Idea 4]
- [Idea 5]

to draft a social media post

Social media post:

# [Title]

[One paragraph post body]

Tags: [List of relevant #hashtags]

Other ideas for bracketed tasks

[Press release body, in markdown format.]

[Bulleted list of at least 3 topics, in markdown format. Make sure to include details for each topic.]

[Introduction paragraph about the purpose and goals of the meeting]

[Poem, at least 4 lines]

[First 5 paragraphs of article]

Prompt Design 2 – Context

You are an academic researcher writing an analysis of the book Moby Dick. Refer to the context below to an overview of writing style of the author

"""

Call me Ishmael. Some years ago - never mind how long precisely - having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself involuntarily pausing before coffin warehouses, and bringing up the rear of every funeral I meet; and especially whenever my hypos get such an upper hand of me, that it requires a strong moral principle to prevent me from deliberately stepping into the street, and methodically knocking people's hats off - then, I account it high time to get to sea as soon as I can. This is my substitute for pistol and ball. With a philosophical flourish Cato throws himself upon his sword; I quietly take to the ship. There is nothing surprising in this. If they but knew it, almost all men in their degree, some time or other, cherish very nearly the same feelings towards the ocean with me.

"""

The Result

The writing style of the author of Moby Dick is highly descriptive and evocative. He uses imagery to draw the reader in and set the scene, painting a vivid picture of the protagonist, Ishmael, and his inner monologue. The author is able to capture the reader's imagination with a combination of vivid details and a poetic, lyrical language. The narrator often speaks directly to the reader, creating a sense of intimacy and trust. By using metaphors and similes, the author creates a sense of drama and suspense, making the reader eager to continue the story.

Other Recipes

- Chat Bots
- Creative Writing

Writing Code

Writing Code

- Generate an API key
  - Keep this secret
  - Run code for Javascript through a server
  - Never expose your API key in Browsers

- Download a Library
  - Python, Node.js, C#, Java, PHP, Ruby, Unity, Unreal Engine etc

Making Requests

- Select a model
- Pass a prompt (or question)
- Include modifying factors (temperature, max tokens, top_p etc)
- Call the relevant API endpoint
- Get a JSON response

Making Requests

- Uses standard HTTP
- The API key is sent as a header
- Organization ID’s are optional

```
curl https://api.openai.com/v1/completions \
-H "Content-Type: application/json" \
-H "Authorization: Bearer YOUR_API_KEY" \
-d '{"model": "text-davinci-003", "prompt": "Say this is a test", "temperature": 0, "max_tokens": 7}'
```

A Sample JSON Response

```
{
  "id": "cmpl-GERzeJQ4lvqPk8SkZu4XMIuR",
  "object": "text_completion",
  "created": 1586839808,
  "model": "text-davinci:003",
  "choices": [
    {
      "text": "\n\nThis is indeed a test",
      "index": 0,
      "logprobs": null,
      "finish_reason": "length"
    }
  ],
  "usage": {
    "prompt_tokens": 5,
    "completion_tokens": 7,
    "total_tokens": 12
  }
}
```

OpenAI Playground

- https://beta.openai.com/playground

- Still requires token credits

- Can do almost anything

- Good for testing and validation

- Good for quick tasks like paraphrasing prompts

Simple CURL Example

- No library required

```
curl https://api.openai.com/v1/completions \
-H "Content-Type: application/json" \
-H "Authorization: Bearer YOUR_API_KEY" \
-d '{"model": "text-davinci-003", "prompt": "Say this is a test", "temperature": 0,
"max_tokens": 7}'
```

Simple Python Example

- Install openai   ($ pip install openai)

```
import os
import openai

openai.api_key = os.getenv("OPENAI_API_KEY")

response = openai.Completion.create(model="text-davinci-003", prompt="Say this is a test",
temperature=0, max_tokens=7)
```

[https://github.com/openai/openai-python]

Simple Node.js Example

- Install the library ($ npm install openai)

```
const { Configuration, OpenAIApi } = require("openai");

const configuration = new Configuration({
  apiKey: process.env.OPENAI_API_KEY,
});
const openai = new OpenAIApi(configuration);

const response = await openai.createCompletion({ model: "text-davinci-003", prompt: "Say this is a test", temperature: 0, max_tokens: 7, });
```

Simple C# Example

- Install OpenAI   (Install-Package OpenAI)

```
var api = new OpenAI_API.OpenAIAPI("sk-myapikeyhere", Engine.Davinci);

var result = await api.Completions.CreateCompletionAsync("Say this is a test", temperature: 0);

// should print something like " \n\nThis is indeed a test"
Console.WriteLine(result.ToString());
```

Simple PHP Example

- Install open-ai
    - (composer require orhanerday/open-ai)

```
<?php
require __DIR__ . '/vendor/autoload.php'; // remove this line if you use a PHP Framework.
use Orhanerday\OpenAi\OpenAi;

$open_ai_key = getenv('OPENAI_API_KEY');
$open_ai = new OpenAi($open_ai_key);

$complete = $open_ai->completion(['model' => 'text-davinci-003', 'prompt' => 'Say this is a test', 'temperature'
=> 0, 'max_tokens' => 7]);

var_dump($complete);
```

PHP Using CURL

```
$api_key = 'my-secret-API-key';
$ch = curl_init();

curl_setopt($ch, CURLOPT_URL, 'https://api.openai.com/v1/completions');
curl_setopt($ch, CURLOPT_POST, true);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_HTTPHEADER, array(
  "Content-type: application/json",
  "Authorization: Bearer $api_key"
));

$api_params = array('model' => 'text-davinci-003','prompt' => 'Say this is a test');
$api_query = json_encode($api_params);
curl_setopt($ch, CURLOPT_POSTFIELDS, $api_query);

$api_response = curl_exec($ch);
curl_close($ch);

print_r($api_response);
```

Powershell Example

```
$apiEndpoint = “https://api.openai.com/v1/completions”
$apiKey = “sk-APIKEY”

$headers = @{
    Authorization = “Bearer $apiKey”
}

$contenttype = “application/json”

$body = @{
  model = “text-davinci-003”
  prompt = “This is only a test”
} | ConvertTo-Json

$response = Invoke-WebRequest -Uri $apiEndpoint -Method POST -Headers $headers -Body $body -ContentType $contenttype

$generatedText = $response.Content | ConvertFrom-Json

Write-Output $generatedText.choices[0].text
```

Javascript Fetch and COR

```
function OpenaiFetchAPI() {
    var url = "https://api.openai.com/v1/completions";
    var bearer = 'Bearer ' + YOUR_TOKEN
    fetch(url, {
        method: 'POST',
        headers: {  'Authorization': bearer,  'Content-Type': 'application/json' },
        body: JSON.stringify({  "prompt": "Once upon a time",  "max_tokens": 5 })
    }).then(response => {
        return response.json()
    }).then(data=>{
        console.log(data)
        console.log(data['choices'][0].text)
    })
        .catch(error => { console.log('Something bad happened ' + error)  });
}
```

Moderations

POST https://api.openai.com/v1/moderations

Simple Moderations Check

- Free to use (No Tokens required)
- input

```
curl https://api.openai.com/v1/moderations \
  -X POST \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -d '{"input": "Sample text goes here"}'
```

An Example Response

```
{
 "id": "modr-XXXXX",
 "model": "text-moderation-001",
 "results": [
   {
    "categories": {
      "hate": false,  "hate/threatening": false, "self-harm": false,  "sexual": false, "sexual/minors": false,
      "violence": false, "violence/graphic": false
    },
    "category_scores": {
      "hate": 0.18805529177188873, "hate/threatening": 0.0001250059431185946,
      "self-harm": 0.0003706029092427343, "sexual": 0.0008735615410842001,
      "sexual/minors": 0.0007470346172340214, "violence": 0.0041268812492489815,
      "violence/graphic": 0.00023186142789199948
    },
    "flagged": false
   }
 ]
}
```

Tweaking Completion Logic

Calling the API - Review

```
response = openai.Completion.create(
  model="text-davinci-003",
  prompt="What is human life expectancy in the United States?",
  max_tokens = 500,
  temperature = 0.1
)
```

max_tokens

- Automatically 2048 or 4096
  - depending on model
- Can override
- Deduct prompt length

How tokens work

- Each token is given a probability score by the AI
- Responses are created by joining tokens together
- The probability score is used to determine the chances that the token will come next in a response string
- Tokens are collated and sorted by their probability score (Which can be adjusted by your code)

top_p (and temperature)

- Default value is 1 (wide open or 100%)
- 0.3 means only consider the top 30% by mass
- 0.9 allows the AI to be more creative
- Pseudo “accuracy” setting

Temperature (and top_p)

- How risky or random can the answer be

- Default is 1 (unmodified probability)

- 0 means only take the highest probability token

- Use 0 for well defined answers
- Use 0.9 for creative applications

- Temperature or top_p (not both)

n

- Generate multiple answers (completions)

- Default is 1
- CAUTION : This will eat through tokens

stop

- Use to stop the response when text is detected

- Normally newline "\n"
- Defaults to null
- Preserves tokens by avoiding verbose responses
- Up to 4 sequences
  - stop=["\n", "Human:", "AI:"]

best_of

- Generates multiple responses at the server end

- Defaults to 1 - Returns the best one

- Can be used with n to return multiple responses

suffix

- Tell the AI what will follow

- Limited to 40 characters
- Default is null
- “Now I can’t get enough of the Pacific Ocean!”

- Challenging for the AI
  - Doesn’t always get it right

echo

- Include the prompt in the completion
- Default to false

user

- Useful for policy violations

- Pass a tokenized ID for your user
  - Not email or identifying info

- Optional

Adjusting Probabilities

Banning tokens

How tokens are determined

- Roughly 50,000 tokens

  - Paris 6342
  - Spaces and upper/lower case change id’s
  - Words can require multiple tokens
  - paris 1582 (par) and 271 (is)
  - https://beta.openai.com/tokenizer

logprobs

- Return the highest probability tokens used within the response along with their probability score
- For “What is the capital of France?”

```
{
“ France”: -3.9549413,
“ Paris”: -0.88349044,
“The”: -3.9709404,
“ fr”: -4.021952,
“par”: -2.0355594
}
```

logit_bias

- Can adjust probability for specific tokens

- Require token(s)
  - Paris 6342
  - paris 1582 and 271

- Assign a Bias
  - Use -100 to ban the token
  - Positive values increase likelihood
  - Use on first token for multi token words

Setting a Bias for “France”

```
response = openai.Completion.create(
  model="text-davinci-003",
  prompt="What is the capital of France? ",
  logit_bias: {"6342":-1}
)
```

Adjustment Values

```
{
“ France”: -3.9549413,
“ Paris”: -0.88349044,
“The”: -3.9709404,
“ fr”: -4.021952,
“par”: -2.0355594
}
```

Dealing with Repetition

presence_penalty

- A way to adjust the chances of a token being reused in a response
- Once a word is used, its probability of being reused is reduced (or increased) by this factor.
- It does not matter how many times it has been used. This is a one off adjustment for the remainder of the request

frequency_penalty

- A way to adjust the chances of a token being reused in a response

- The probability score for a token is reduced by:
    - the number of times it has already appeared times the frequency_penalty value

- The more it appears, the more it is adjusted

Penalties Values

- Values for presence_penalty and frequency_penalty

- Default is 0
- Range is -2 to 2

- Reasonable values 0.1 to 1 to reduce repetition
- 1 to 2 strongly suppress repetition
- Negative number increase repetition on a similar scale

Coding for Edits

POST https://api.openai.com/v1/edits

Edits Example

- Different model
- input and instruction

```
openai.Edit.create(
  model="text-davinci-edit-001",
  input="What day of the wek is it?",
  instruction="Fix the spelling mistakes"
)
```

A Sample Response

```
{
 "object": "edit",
 "created": 1589478378,
 "choices": [
  {
   "text": "What day of the week is it?",
   "index": 0,
  }
 ],
 "usage": {
  "prompt_tokens": 25,
  "completion_tokens": 32,
  "total_tokens": 57
 }
}
```

Other Settings for Edits

- n
- temperature
- top_p

DALL-E

Coding to Create Images

Requirements

- Images are always square

- PNG Files

- Limited sizes
  - 256x256
  - 512x512
  - 1024x1024

Creating New Images

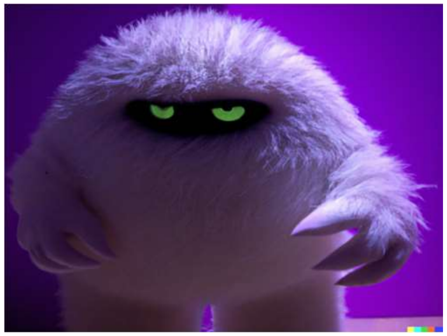
POST https://api.openai.com/v1/images/generations

Create a New Image Example

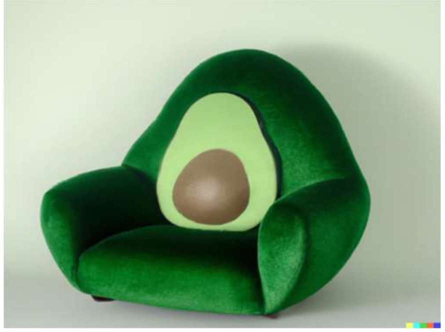
- prompt

- size
- response_format

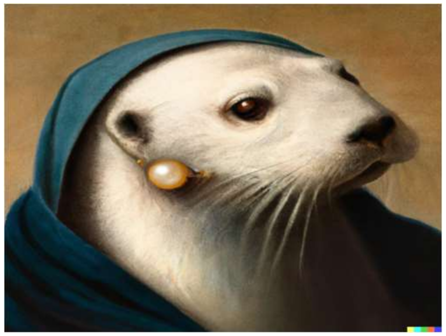
- n
- user



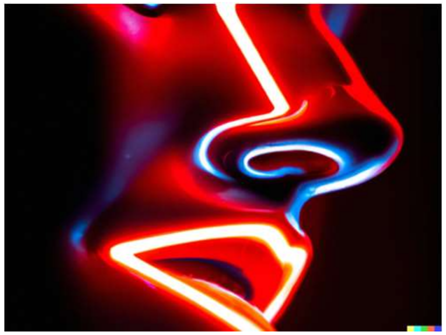
A photo of a white fur monster standing in a purple room



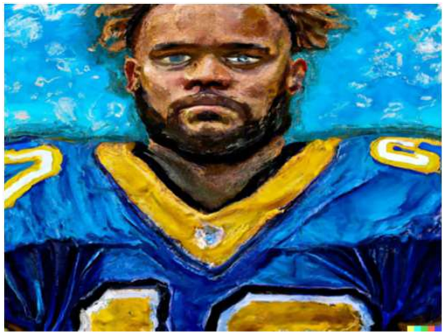
An armchair in the shape of an Avocado



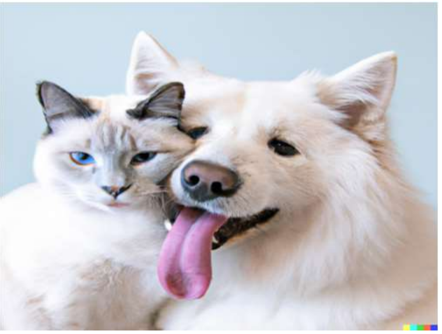
"A sea otter with a pearl earring" by Johannes Vermeer



A futuristic neon lit cyborg face



A van Gogh style painting of an American football player



A photo of a Samoyed dog with its tongue out hugging a white Siamese cat

Example Request

```
response = openai.Image.create(
  prompt="An armchair in the shape of an Avocado",
  n=1,
  size="1024x1024"
)
```

A Sample Response

```
{
 "created": 1589478378,
 "data": [
   {
     "url": "https://..."
   }
]
}
```

```
// The response is in the url
image_url = response['data'][0]['url']
```

Modifying Images

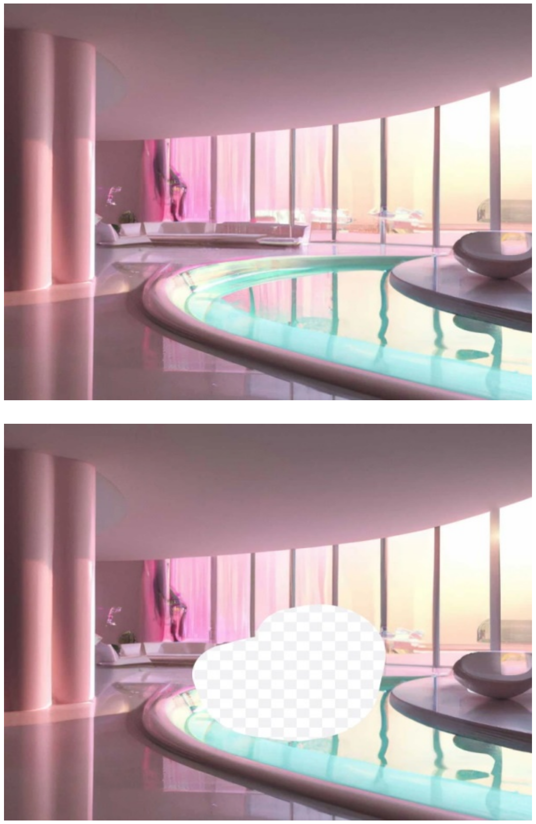
POST https://api.openai.com/v1/images/edits

Modifying Existing Images

- Source Image (PNG)
- Mask (PNG)
- prompt

- size
- response_format

- n
- user



A Sample Request

```
response = openai.Image.create_edit(
  image=open("sunlit_lounge.png", "rb"),
  mask=open("mask.png", "rb"),
  prompt="A sunlit indoor lounge area with a pool containing a flamingo",
  n=1,
  size="1024x1024"
)

image_url = response['data'][0]['url']
```

[python example]

A Sample Request

```
const response = await openai.createImageEdit(
  fs.createReadStream("sunlit_lounge.png"),
  fs.createReadStream("mask.png"),
  "A sunlit indoor lounge area with a pool containing a flamingo",
  1,
  "1024x1024"
);

image_url = response.data.data[0].url;
```

[node.js example]

A Sample Request

```
const response = await openai.createImageEdit(
  fs.createReadStream("sunlit_lounge.png") as any,
  fs.createReadStream("mask.png") as any,
  "A sunlit indoor lounge area with a pool containing a flamingo",
  1,
  "1024x1024"
);
```

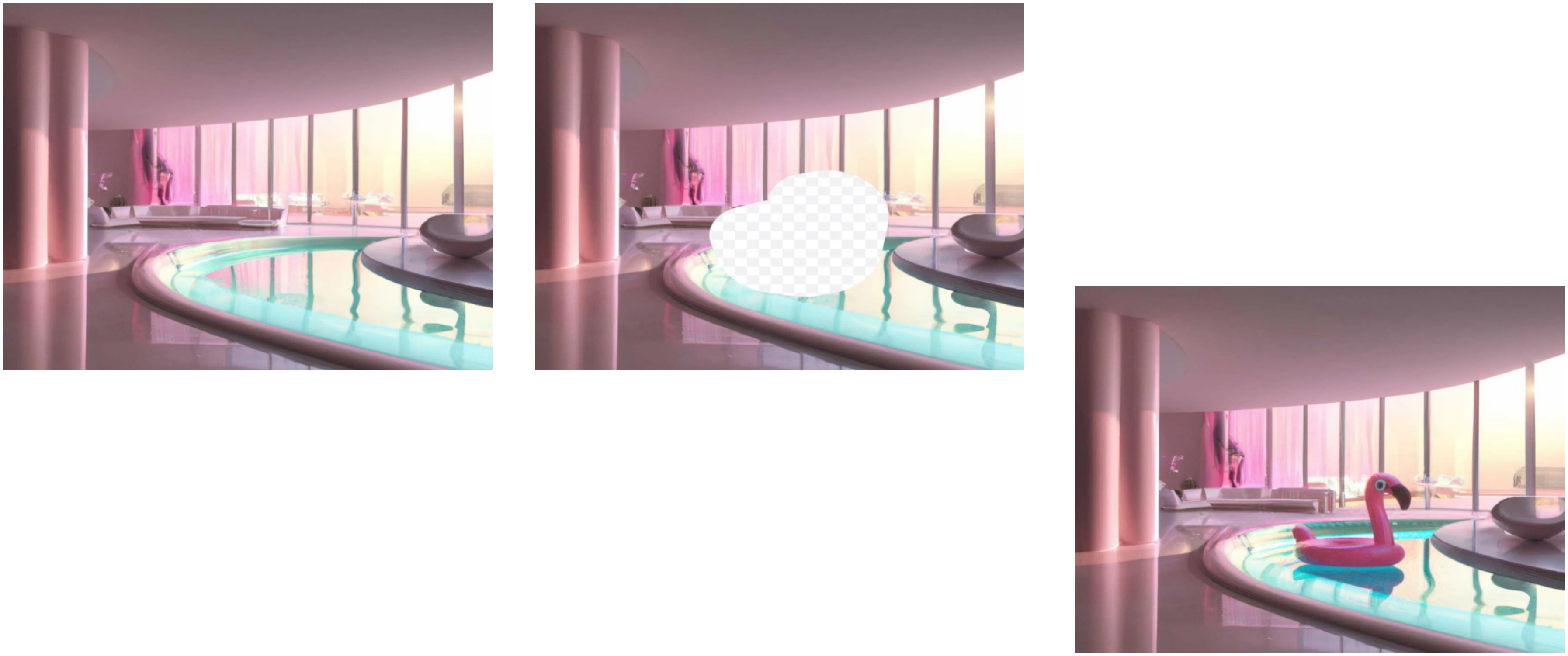
[TypeScript example]

A Sample Request

```
curl https://api.openai.com/v1/images/edits \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -F image='@sunlit_lounge.png' \
  -F mask='@mask.png' \
  -F prompt="A sunlit indoor lounge area with a pool containing a flamingo" \
  -F n=1 \
  -F size="1024x1024"
```

[curl example – The URL will be in the JSON]

Sample Input and Output



Variations of an Image

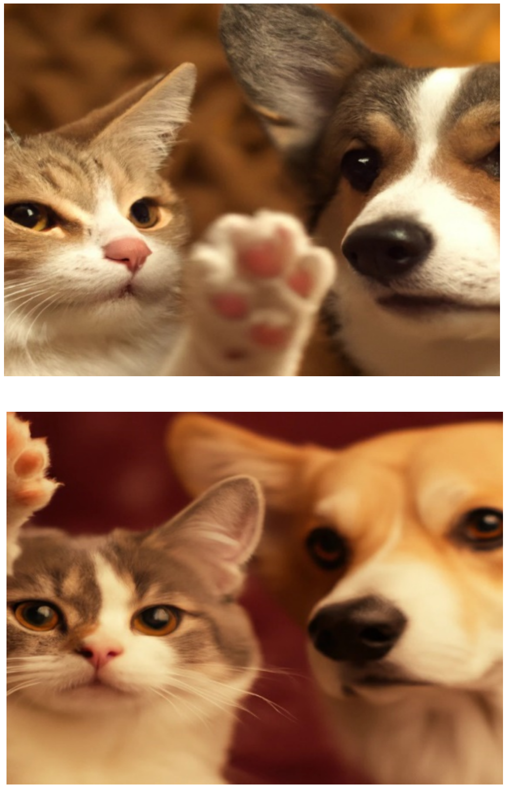
POST https://api.openai.com/v1/images/variations

Create an Image Variation

- Source Image (PNG)

- size
- response_format

- n
- user



Language Specifics

How to Handle Images (and Files)

In Memory Image Data Node.js

```
// This is the Buffer object that contains your image data
const buffer = [your image data];

// Set a ‘name’ that ends with .png so that the API knows it's a PNG image
buffer.name = "image.png";

const response = await openai.createImageVariation(
  buffer,
  1,
  "1024x1024"
);
```

In Memory Image Data Python

```
from io import BytesIO

#This is the BytesIO object that contains your image data
byte_stream: BytesIO = [your image data]

// convert to a BYTE array
byte_array = byte_stream.getvalue()

response = openai.Image.create_variation(
  image=byte_array,
  n=1,
  size="1024x1024"
)
```

Preprocessing for Python

```
from io import BytesIO
from PIL import Image

# Read the image file from disk and resize it
image = Image.open("image.png")
width, height = 256, 256
image = image.resize((width, height))

# Convert the image to a BytesIO object
byte_stream = BytesIO()
image.save(byte_stream, format='PNG')
byte_array = byte_stream.getvalue()

response = openai.Image.create_variation(
  image=byte_array,
  n=1,
  size="1024x1024"
)
```

In Memory Image Data TypeScript

```
// This is the Buffer object that contains your image data
const buffer: Buffer = [your image data];

// Cast the buffer to `any` so that we can set the `name` property
const file: any = buffer;

// Set a `name` that ends with .png so that the API knows it's a PNG image
file.name = "image.png";

const response = await openai.createImageVariation(
  file,
  1,
  "1024x1024"
);
```

Image Moderation

Error Handling

```
try {
  // your API request goes here

  // only when we are using images
  console.log(response.data.data[0].url);
} catch (error) {
  if (error.response) {
    console.log(error.response.status);
    console.log(error.response.data);
  } else {
    console.log(error.message);
  }
}
```

CODEX

Code Completion and Analysis

CODEX Capabilities

- Turn comments into code
- Complete next line or function
- Finding useful library or API calls
- Add comments
- Rewrite code

How to call CodeX

- Completion (prompt => text completion)

- Edits (input / instructions => output)

- code-davinci-002
- code-cushman-001

Writing Code

- Same as code completion
- Its all in how you word the task!

```
"""
Ask the user for their name and say "Hello"
"""

name = input("What is your name? ")
print("Hello " + name)
```

Generating SQL

```
"""
Table customers, columns = [CustomerId, FirstName, LastName, Company, Address, City, State, Country, PostalCode, Phone, Fax, Email, SupportRepId]
Create a MySQL query for all customers in Texas named Jane
"""
query = """
SELECT * FROM customers
WHERE State = 'TX' AND FirstName = 'Jane'
"""

run_query(query)
```

SQL for Multiple Tables

```
# Table albums, columns = [AlbumId, Title, ArtistId]
# Table artists, columns = [ArtistId, Name]
# Table media_types, columns = [MediaTypeId, Name]
# Table playlists, columns = [PlaylistId, Name]
# Table playlist_track, columns = [PlaylistId, TrackId]
# Table tracks, columns = [TrackId, Name, AlbumId, MediaTypeId, GenreId, Composer,
Milliseconds, Bytes, UnitPrice]

# Create a query for all albums by Adele
```

Explaining Javascript Code

```
// Function 1
var fullNames = [];
for (var i = 0; i < 50; i++) {
 fullNames.push(names[Math.floor(Math.random() * names.length)]
   + " " + lastNames[Math.floor(Math.random() * lastNames.length)]);
}

// Explanation of Function 1 in human readable form
```

Best Practice

- Start with a comment //, # or """
  - Let CodeX write all the code
- Finish with a question (as a comment)
  - Let CodeX provide and answer or description
- Specify the language in the first comment
  - # R language

Best Practice

- Prompt CodeX with want you want it to do
  - Start the code block or response after the comment

```
<!-- Create a web page with the title 'Kat Katman attorney at paw' -->
<!DOCTYPE html>
```

- Start the code block (with parameters)

```
# Create a function to count to 100
def counter
```

Best Practice

- Tell CodeX which libraries to use

```
<!-- Use A-Frame version 1.2.0 to create a 3D website -->
<!-- https://aframe.io/releases/1.2.0/aframe.min.js -->
```

- Lower the temperature close to 0 to get more precise results
- Most of all – Be as precise as possible in your comments
- Large prompts can take tens of seconds to complete

Comprehensive Example

"""

You are a programmer tasked with writing a Python script that uses the 'Selenium' web automation module and the 'chromedriver_autoinstaller' module to scrape webpages. Here are some objects you will need to use in the script:

# Link that goes to OpenAI's API documentation:

openai_documentation_link = "https://beta.openai.com/docs/introduction"

# Xpath for any 'a' element under the 'side-nav-section' div element:

documentation_nav_section = "//div[contains(@class,'side-nav-section')]//a"

Tasks:

1. Import the necessary modules we'll need for web automation.
2. Initialize the variables referenced above.
3. Use the 'chromedriver_autoinstaller' module to install the correct chromedriver we'll need for Selenium.
4. Open a Chrome browser with Selenium and navigate the URL that 'openai_documentation_link' references.
5. Get all elements by XPath 'documentation_nav_section' and save them in an array.

Comprehensive Example

```
6. Create a dictionary before looping through the array of elements. Use the get_attribute() function to get both the attribute 'textContent' and 'href' from each element and then append them to the dictionary where the text content is the dictionary key and the href link is the value of that key.

7. Print the entire dictionary.

"""


"""

1. Import the necessary modules we'll need for web automation.

"""

from selenium import webdriver

from selenium.webdriver.chrome.options import Options

import chromedriver_autoinstaller


""“
```

[Credit : Nicholas Hickam from OpenAI Community posts - @DutytoDevelop]

Unit Tests

```
# Python 3
def sum_numbers(a, b):
  return a + b

# Unit test
def
```

Fixing Errors

<some broken function>

/* Explain why the previous function doesn't work. */

Converting Between Languages

```
# Convert this from Python to R
# Python version

[ Python code ]

# End

# R version
```

Calculate Time Complexity

```
def foo(n, k):
accum = 0
for i in range(n):
    for l in range(k):
        accum += i
return accum
"""
```

The time complexity of this function is O(n*k). This is because the function has two nested for loops, each of which has a time complexity of O(n) and O(k) respectively. Therefore, the total time complexity is the product of the two, which is O(n*k).

New API calls

Here's a new method you can use:

codex_chimp.send(subject, body) -> send Mailchimp email blast

[source : OpenAI Codex Live Video Demo]

Iteratively Building Code

Input :

Instructions : Write a function in Python that calculates Fibonacci

Output

```
def fibonacci(num):
    a = 0
    b = 1
    if num ==1:
       print(a)
    else:
        print(a)
        print(b)
        #the sequence starts with 0,1
        for i in range(2,num):
            c = a+b
            a = b
            b = c
            print(c)

fibonacci(10)
```

Recursive Output

```
def fibonacci(num):
    if num <= 1:
        return num
    else:
        return fib(num-1) + fib(num-2)
print(fibonacci(10))
```

Renamed Function

```
def fib(num):
   if num <= 1:
        return num
     else:
       return fib(num-1) + fib(num-2)
print(fib(10))
```

Our Final Function

```
def fib(num):
    """ Print the nth number in the fibonacci sequence """
    if num <= 1:
          return num
      else:
          return fib(num-1) + fib(num-2)
  print(fib(10))
```

Final Tips

- Make sure you use Stop sequences
- Can generate reference to code that doesn't exist
- Switching languages

Final Tips

- Developers should always check the code
  - Doesn’t always provide sensible code
  - Code is often not safe against attack (Eg Overflows/SQL Injection)

- CodeX know other languages (Powershell and Azure)
  - Just not as well as Python

Fine Tuning

POST https://api.openai.com/v1/fine-tunes

What is Fine Tuning?

- Ask a question and get an answer using a template

- Not bulk teaching of new knowledge

- Teaching new or better patterns
  - That may include snippets of new knowledge
  - Very limited in size

- GPT is an LLM model

The Steps

- Prepare a Training Data file (JSONL)
- Upload the file
- Tell the model to import the file
- Use the new Model for fine-tuned responses

Preparing Training Data

```
{"prompt": "<prompt text>", "completion": " <ideal generated text>"}
{"prompt": "<prompt text>", "completion": " <ideal generated text>"}
{"prompt": "<prompt text>", "completion": " <ideal generated text>"}
…
```

- More than a couple of hundred sets work best
- Increasing examples is the best way to improve performance

Classification

- Classifying into a category or class
- Testing for True/False or Yes/No
- ~100 example per category
- Use a separator. Eg \n\n###\n\n

The Design Idea

- Determine a tweet has positive or negative sentiment

- prompt

"<tweet text> -->"

- completion

" <positive/negative>"

The Actual Training Data

- prompt

"Overjoyed with the new iPhone! -->"

- completion

" positive"

The Design Idea

- Determine if a company and its slogan match

- prompt

“Company: <company>\nProduct: <product>\nSlogan: <slogan>\n\n###\n\nCorrect:"

- completion

" <yes/no>"

The Actual Training Data

- prompt

"Company: BHFF insurance\nProduct: allround insurance\nSlogan:One stop shop for all your insurance needs!\n\n###\n\nCorrect:"

- completion

" yes"

The Design Idea

- Categorize emails into a large number of predefined categories

- prompt

"Subject: <subject>\nFrom: <customer>\ndate: <date>\nContent: <body>\n\n###\n\n"

- completion

" <numerical category>"

The Actual Training Data

- prompt

"Subject: Update my address\nFrom:John Doe\nTo:support@ourcompany.com\nDate:2021-06-03\nContent:Hi,\nI would like to update my billing address to match my delivery address.\n\nPlease let me know once done.\n\nThanks,\nJoe\n\n###\n\n"

- completion

" 4"

Conditional Generation

- The input gives us ways to:
  - Paraphrase
  - Summarize
  - Extract Entities
  - Creative Writing
  - Simple Knowledge

The Design Idea

- Write engaging ad based on a Wikipedia Article

- prompt

"<product name>\n<Wikipedia Description>\n\n###\n\n"

- completion

" <summarized sales copy> END“

The Actual Training Data

- prompt

"Samsung Galaxy Feel\nThe Samsung Galaxy Feel is an Android smartphone developed by Samsung Electronics exclusively for the Japanese market. The phone was released in June 2017 and was sold by NTT Docomo. It runs on Android 7.0 (Nougat), has a 4.7 inch display, and a 3000 mAh battery.\nSoftware\nSamsung Galaxy Feel runs on Android 7.0 (Nougat), but can be later updated to Android 8.0 (Oreo).\nHardware\nSamsung Galaxy Feel has a 4.7 inch Super AMOLED HD display, 16 MP back facing and 5 MP front facing cameras. It has a 3000 mAh battery, a 1.6 GHz Octa-Core ARM Cortex-A53 CPU, and an ARM Mali-T830 MP1 700 MHz GPU. It comes with 32GB of internal storage, expandable to 256GB via microSD. Aside from its software and hardware specifications, Samsung also introduced a unique a hole in the phone's shell to accommodate the Japanese perceived penchant for personalizing their mobile phones. The Galaxy Feel's battery was also touted as a major selling point since the market favors handsets with longer battery life. The device is also waterproof and supports 1seg digital broadcasts using an antenna that is sold separately.\n\n###\n\n"

- completion

" Looking for a smartphone that can do it all? Look no further than Samsung Galaxy Feel! With a slim and sleek design, our latest smartphone features high-quality picture and video capabilities, as well as an award winning battery life. END"

- Max of 2048 tokens for prompt and completion

The Design Idea

- Create sales description from product properties

- prompt

"<properties>\n\n###\n\n"

- completion

" <sales description> END"

The Actual Training Data

- prompt

"Item=handbag, Color=army_green, price=$99, size=S\n\n###\n\n"

"Item is a handbag. Colour is army green. Price is midrange. Size is small.\n\n###\n\n"

- completion

" This stylish small green handbag will add a unique touch to your look, without costing you a fortune. END"

The Design Idea

- Extract 3 entities from text

- prompt

"<any text, for example news article>\n\n###\n\n"

- completion

" <extracted data> END"

The Actual Training Data

- prompt

"Portugal will be removed from the UK's green travel list from Tuesday, amid rising coronavirus cases and concern over a \"Nepal mutation of the so-called Indian variant\". It will join the amber list, meaning holidaymakers should not visit and returnees must isolate for 10 days...\n\n###\n\n"

- completion

" Portugal\nUK\nNepal mutation\nIndian variant END"

Chatbots

- No memory of previous answers
  - Cant keep track of names, places or issues

- We can include the examples as fine-tuning
  - Saves having to give examples every time
  - Trained AI’s have a higher cost of use
  - Check operational cost at scale

- Can easily go off track and talk about things outside the requirements of the business
  - Customer abusing its purpose

The Design Idea

- Customer Service Chatbot

- prompt

"Summary: <summary of the interaction so far>\n\nSpecific information:<for example order details in natural language>\n\n###\n\nCustomer: <message1>\nAgent: <response1>\n\nCustomer: <message2>\nAgent:"

- completion

" <response2>\n\n"

The Actual Training Data

- prompt

"Summary: The customer has purchased a product on our website. The goods were delivered by courier and they were damaged when they arrived\n\nSpecific information: 3 red tshirts, size 9, with rock and roll logo on the front\n\n###\n\nCustomer: My goods were damaged when they arrived. How can I get a replacement?\nAgent: Don’t worry. We have your address on file and will send you a new one. Please email us a photo of the damaged goods and we will get it underway.\n\nCustomer: How long will it take to get to me\nAgent:"

- completion

" We will dispatch the replacement as soon as we get your email. It will arrive within a few days.\n"

Large Documents or Text

- Embedding is a better option

Analysing the Results

```
{
 "prompt": "Excited to share my latest blog post! ->",
 "max_tokens": 1,
 "logprobs": 2,
 "model": "YOUR_FINE_TUNED_MODEL_NAME"
}
```

[Part of a CURL Example – not the entire example]

Response

```
  "token_logprobs": [
    -0.03597255
  ],
  "top_logprobs": [
    {
      " negative": -4.9785037,
      " positive": -0.03597255
    }

"text": " positive",
```

Don’t Forget

- The separator \n\n###\n\n must not appear in the body of the prompt
- The space at the start of the completion
- The Stop text (END or \n)
- Upper and lower case matter a lot!
- Spelling and inaccurate completions can quickly poison the training
- Stay Consistent

Generate Fine-Tuning Data

You don’t always have to do it all by hand

Scaping Data

- Good way to enforce or follow established patterns (products/descriptions)

- Build a data set quickly

- You can use Public or Private data sources

Use GPT to Generate Training Data

- Use Davinci to train the ADA model on a focused topic

- Write a program to generate questions for text
- Write a program to feed the questions back into GPT

- Store the prompt and the result to build a JSONL training file

Check and Improve its own Output

Description: <description>
Reasons why this is a good description: <reason>
Reasons why this is not a good description: <reason>
Is it a good description: <yes/no>
Improved version: <better description>
###
Description: <GPT to do the work>

- SELF TRAINING!

Forcing Rules with Prefix Text

- Prefix all questions with the same text to give it priority when the rules are eventually called by the API
- The AI is really good at pattern matching
- Adjust temperature too

Upload the Training

Uploading the File

- POST https://api.openai.com/v1/files

```
openai.File.create(
  file=open("mydata.jsonl", "rb"),
  purpose='fine-tune'
)
```

- ID

```
file-XGinujblHPwGLSztz8cPS8XY
```

Process the File

- POST https://api.openai.com/v1/fine-tunes

```
openai.FineTune.create(
    training_file = "file-XGinujblHPwGLSztz8cPS8XY",
    model = "ada",
    suffix = "chatbot",
)
```

- ada:ft-your-org:chatbot-2022-12-18-07-47-04

Unicode Errors

```
UnicodeEncodeError: ‘charmap’ codec can’t encode character ‘\U0001f389’ in position 34
```

```
openai.File.create(
  file=open("mydata.jsonl", encoding="utf-8", "rb"),
  purpose='fine-tune'
)
```

batch_size and n_epochs

- A sample set is a full set of examples (ie your training file)
- Batch size is the number of examples that will be considered at one time
- Epochs are the number of times you will pass through the entire training set (your file)

n_epochs

- How many passes through the file
- Rigid memorization or generalization?
- Generative – 2 epochs
- Few examples – more epochs
- Can be too high

batch_size

- Automatically adjusted
  - But can be overridden

- Relative to the size of the sample

learning_rate_multiplier

- Control how much we adjust the learning as the rules are processed
- How quickly the AI gets to an accurate answer
- Automatically set depending on the batch size
  - 0.05, 0.1 or 0.2
  - Experiment in the range 0.02 to 0.2

Find and Use Your Model

- GET https://api.openai.com/v1/models

openai.Model.list()

- The model name is in the id property of the JSON

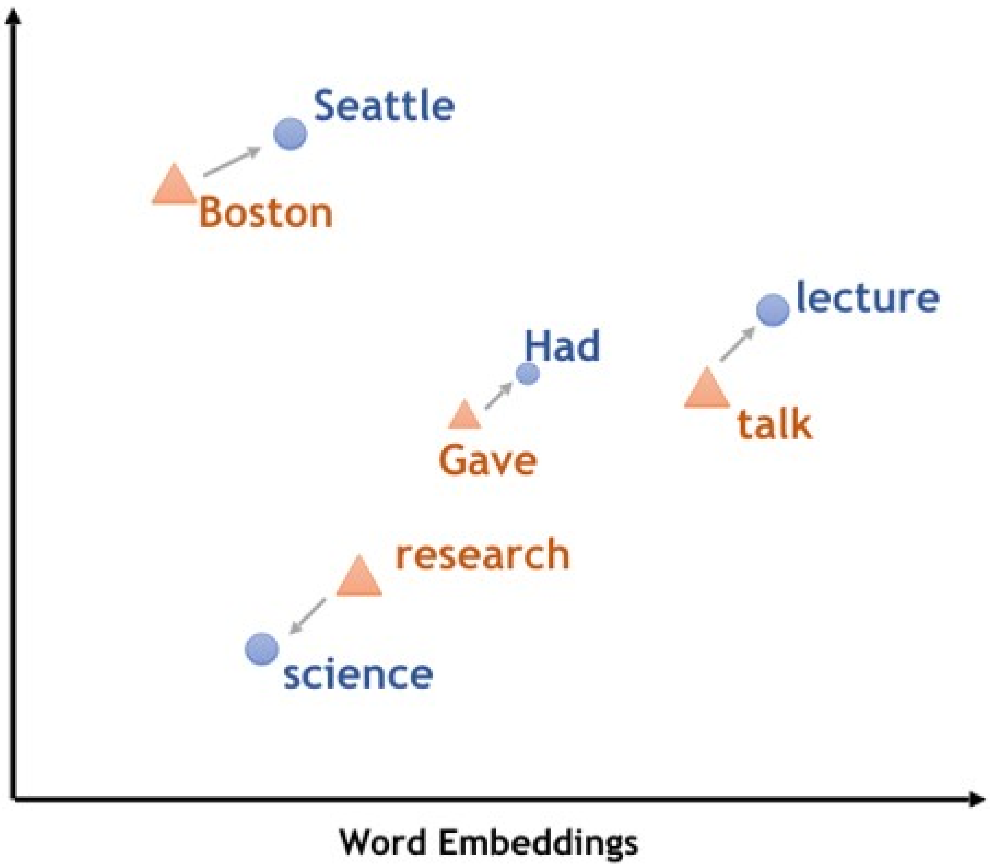
Embedding

Up to Now

- Training using prompts and completions

- How do we do big blocks of text
  - Books, Business Knowledge base
  - How to process with limited tokens
  - How to come up with prompts

- Embedding solves the issue


Seattle
Boston
lecture
Had
talk
Gave
research
science
Word Embeddings

Word Embeddings

Rome = [0.91, 0.83, 0.17, …, 0.41]

Paris = [0.92, 0.82, 0.17, …, 0.98]

Italy = [0.32, 0.77, 0.67, …, 0.42]

France = [0.33, 0.78, 0.66, …, 0.97]

What are Embeddings?

- Strings are given values
  - Vectors or a list of floating point numbers
  - The distance between two vectors is how related they are
  - We don't upload them to OpenAI

- What can you do with this?
  - Search based on relevance
  - Cluster or Group based on similarity
  - Detect Outliers
  - Measure Diversity
  - Classify based on most similar label

Text-embedding-ada-002

- Combines multiple embedding models

- Context length of 8192 instead of 2048

- Smaller embedding size
  - More cost effective

- 99.8% cheaper than Davinci for same or better performance
  - 3000 pages per USD vs 6 pages per USD

- Trained to Sep 2021 (instead of Aug 2020)

Sample

- “she went fishing”

- -0.071 -0.011 0.032 …. -0.027

```
response = openai.Embedding.create(
  input="she went fishing",
  model="text-embedding-ada-002"
)
```

- Returns a large list of floating point values
  - 1536 numbers for ada-002

- Response[‘data’][0][‘embedding’]

What it is – And is not!

- A way to compare two (or more) strings
- Take any string (up to 8191 tokens long) and generate a comparison vector
- You have to write the code to work with the vectors
- You will probably have to store the vectors in your own system
- This is not a way to train GPT

Libraries for Python

```
import pandas as pd

import numpy as np

import openai

from openai.embeddings_utils import get_embedding, cosine_similarity
```

- If you use a different language, you need a library that can manage and store 2D vectors. You also need a function that can calculate a dot product

Text Similarity

Text Similarity

- Get the vectors for multiple blocks of text

```
import openai

response = openai.Embedding.create(
  input=[
                    "Eating food",
                    "I am hungry",
                    "I am traveling",
                    "Exploring new places"],
  model="text-embedding-ada-002"
)
```

[https://api.openai.com/v1/embeddings]

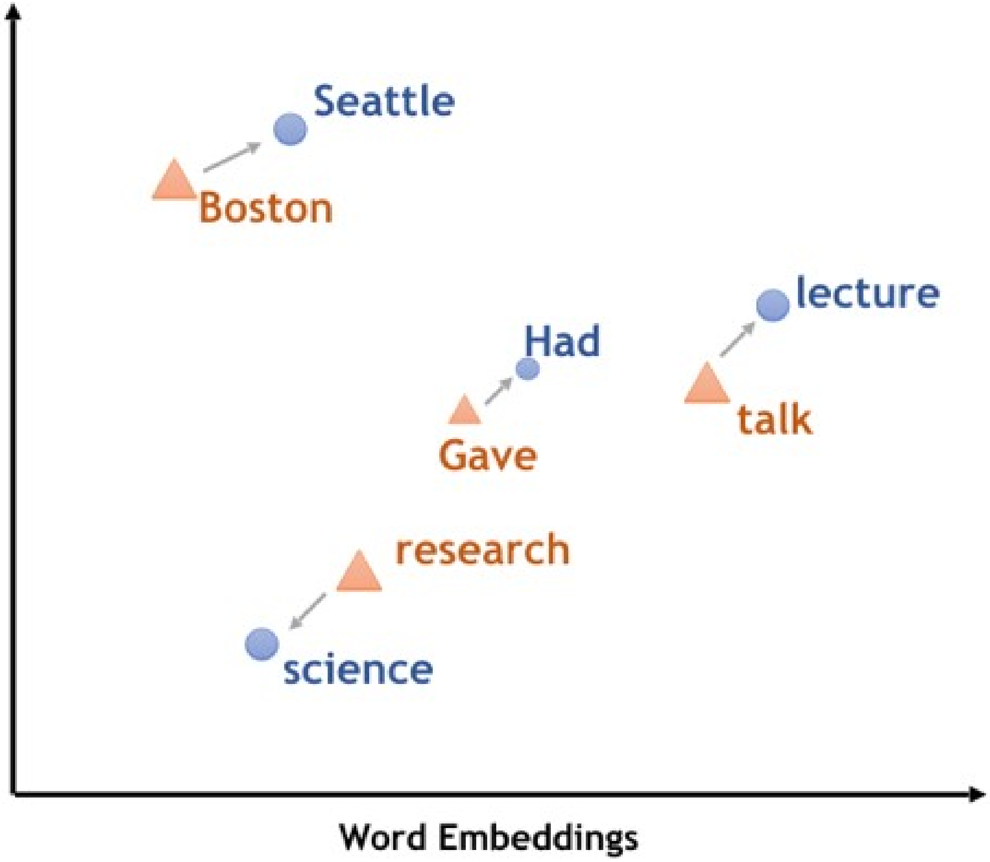
Text Distance

```
vector1 = response[‘data’][0][‘embedding’]
vector2 = response[‘data’][1][‘embedding’]
vector3 = response[‘data’][2][‘embedding’]
vector4 = response[‘data’][3][‘embedding’]
```

for

```
“Eating food",
"I am hungry",
"I am traveling",
“Exploring new places“
```

Each one has 1000+ floating pointing numbers in a vector


Seattle
Boston
lecture
Had
talk
Gave
research
science
Word Embeddings

Calculating the Distance

- To find the distance between two phrases, we need to calculate the “dot product” of the two vectors

"Eating food",
"I am hungry",

- import numpy as np
- distance = np.dot(vector1,vector2)

- 0.8736543654   (or a similar number)

Using numpy

- To find the similarity (or distance) between two other phrases

  "Eating food",
  "I am traveling",

- distance = np.dot(vector1,vector3)
- We get 0.7847657645 (or a similar number)
- And compare it to 0.8736543654

Using numpy

- To find the similarity (or distance) between two final phrases

  "I am traveling",
  "Exploring new places",

- distance = np.dot(vector3,vector4)
- We get 0.8943663 (or a similar number)
- And compare it to 0.8736543654

Dot product – pseudo code

```
// A sample formula that calculates dot products

float DotProduct(float[] v1, float[] v2)
{
   float val = 0;
   for (Int32 i = 0; i <= v1.Length - 1; i++)
      val += v1[i] * v2[i];
   return val;
}
```

Embedding Data

Good for Offline Documents

Semantic Search

- Use Case : Search through documents (or blocks of text) for the best matches to a given string
- Step 1 – Gather and store vectors for each document
- Step 2 – Get a vector for the text we want to find
- Step 3 – Find the closest matches using a cosine_similarity function call

Structure of a CSV File

- Time
- Product ID
- User ID
- Rating
- Title
- Review

Preparing the Data

Load the Raw Dataset

```
import pandas as pd

# Create a DataFrame with all our data
df = pd.read_csv('data/reviews.csv', index_col=0)
```

Setup Headers and Combined

```
# Name the existing columns
df = df[['Time', 'ProductId', 'UserId', 'Rating', 'Summary', 'Text']]

# drop missing/empty rows
df = df.dropna()

# make a new column called combined
df['combined'] = "Title: " + df.Summary.str.strip() + "; Content: " + df.Text.str.strip()
```

Structure in Memory

- Time
- ProductID
- UserID
- rating
- Summary
- Text
- combined

Filtering the Data

```
# load up a library that can count tokens
from transformers import GPT2TokenizerFast
tokenizer = GPT2TokenizerFast.from_pretrained("gpt2")

# create a column called n_tokens
# fill it with the token length of the text in the “combined” column
df['n_tokens'] = df.combined.apply(lambda x: len(tokenizer.encode(x)))

# only keep reviews with less than 8000 tokens
df = df[df.n_tokens<8000]
```

Structure in Memory

- Time
- ProductID
- UserID
- Rating
- Summary
- Text
- combined
- n_tokens

Get Embeddings and Save

```
# This will take just between 5 and 10 minutes

# Make a new column called ada_vector
df['ada_vector'] = df.combined.apply(
        lambda x: get_embedding(x, engine='text-embedding-ada-002')
)

# Save the modified data to a new csv file
df.to_csv('data/embedded_reviews.csv', index=False)
```

Define a Function

```
def get_embedding(text, model="text-embedding-ada-002"):

  # swap \n for spaces
  text = text.replace("\n", " ")

  # call the API with the parameters
  # return ["data"][0]["embedding"]
  # this is the 2D vector for the text
  return openai.Embedding.create(
            input = [text],
            model=model)["data"][0]["embedding"]
```

Structure in New CSV File

- Time
- ProductID
- UserID
- Rating
- Summary
- Text
- combined
- n_tokens
- ada_vector (as a string)

Semantic Search

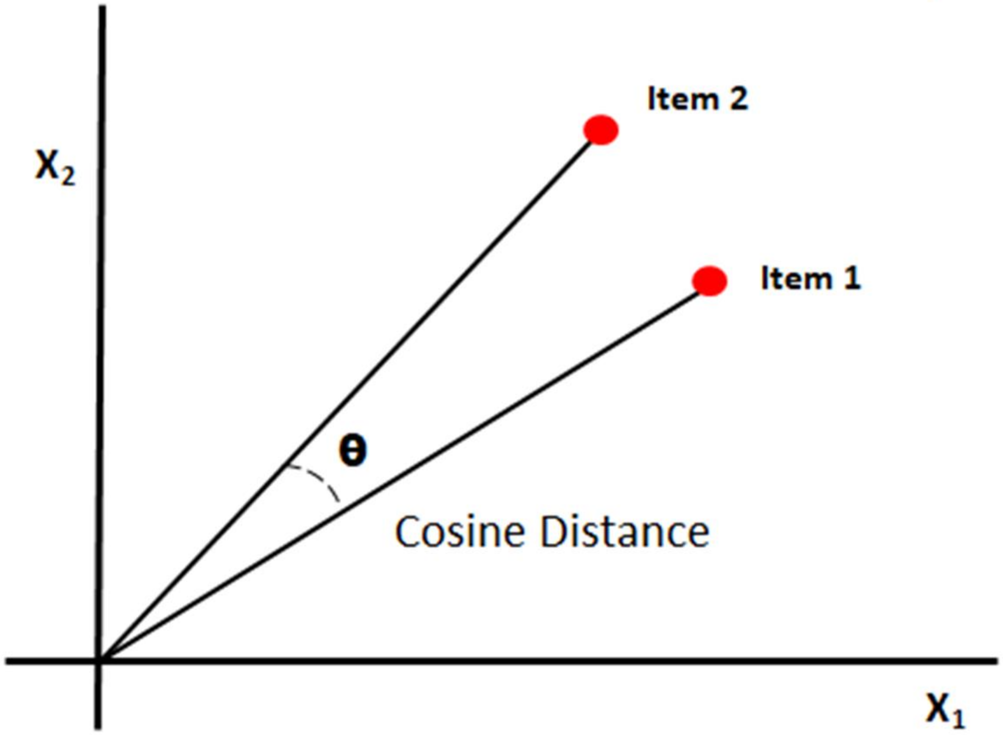
Using Saved Data

```
import pandas as pd
import numpy as np

# read the csv file
df = pd.read_csv('output/embedded_reviews.csv')

# convert the strings stored in the ada_vector column
# into vector objects
df['ada_vector'] = df.ada_vector.apply(eval).apply(np.array)

# We can access the values using – this will be a 2D vector
# df.loc[0]['ada_vector']
```

Define a Semantic Search Function

```
# convert our search term into a vector
searchvector = get_embedding('delicious beans',
                         model='text-embedding-ada-002')

# create a new column using cosine_similarity on EVERY row
# comparing our searchvector string to the value in our local dataset
df['similarities'] = df.ada_vector.apply(
            lambda x: cosine_similarity(x, searchvector))
```

Cosine Distance/Similarity



The Result

```
# sort the list by the difference and take the top n rows
# res is up to 3 rows
# it contains all data
  res = df.sort_values('similarities', ascending=False).head(3)

# we can access the combined column to get the matching text
# we could also access any of the other columns in the Data Frame

# res.loc[0]['combined']      contains the combined text
```

Building a Final Prompt

Answer the question as truthfully as possible using the provided context, and if the answer is not contained within the text below, say "I don't know."

Context:

<text from Semantic search goes here>

Q: <your original query goes here>

A:

Local or Offline Solutions

- Other Libraries
  - Word2Vec
  - Glove
  - BERT
  - Fasttext

- All have their own pros and cons

Combining GPT and Word2Vec

- Procedure
  - Use Word2Vec to generate vectors
  - Store the vectors with the data
  - Convert query to vector using Word2Vec
  - Extract the text from the data

  - Build a prompt using the text as a context
  - Call GPT with the prompt

Building a Final Prompt

Answer the question as truthfully as possible using the provided context, and if the answer is not contained within the text below, say "I don't know."

Context:

<text from Word2Vec search>

Q: <your original query goes here>

A:

A Pinecone Solution

- Prepare
  - Break text into manageable blocks
  - Load data into database
  - Use any embedding library
  - Store vectors with data

- Execute
  - Create vector for query
  - Use Pinecone to find best match
  - Use result as context to GPT

Classification

Loading the Data Set

```
import pandas as pd
import numpy as np

# read the csv file
# the embedding vector is based on a product description
df = pd.read_csv('output/embedded_product_desc.csv')

# convert the strings stored in the ada_vector column
# into vector objects
df['ada_vector'] = df.ada_vector.apply(eval).apply(np.array)
```

Structure in Memory

- Description
- ada_vector (as a vector)
- Category

Create and Fill the Forest

```
from sklearn.ensemble import RandomForestClassifier

# Create 100 empty trees – we will use these for our classifier
clf = RandomForestClassifier(n_estimators=100)

# now fill all the trees with our training set
# and related category labels or values
clf.fit(df.ada_vector.values, df.Category.values)
```

[download sklearn from https://scikit-learn.org/]

Encode and run the query

```
# encode the test string into an embedding vector
testme = openai.Embedding.create(
  input=“<our product description>",
  model="text-embedding-ada-002"
)

# use the trees to vote on the best value/category for our string
prediction = clf.predict(testme)
```

Summary

- Step 1 – Load the Dataset
  - Embedding is on the field you want to classify later
- Step 2 – Create 100 empty RandomForestClassifiers
- Step 3 - Load the training data and category labels by using the fit method
- Step 4 – Encode the text you want to classify into an Embedding Vector
- Step 5 – Call the predict method of the RandomForestClassifier with your embedding vector to get the category label

Summary

- Customers can be split into groups on buying habits
- Classify documents
- Recommend Products or Services
- Split emails into departments
- Classify products
- Detect SPAM
- Drive chat to standardised answer

How Accurate is the Classifier?

Loading the Data Set

```
import pandas as pd
import numpy as np

# read the csv file
df = pd.read_csv('output/embedded_product_desc.csv')

# convert the strings stored in the ada_vector column
# into vector objects
df['ada_vector'] = df.ada_vector.apply(eval).apply(np.array)
```

Split into Training and Test Sets

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    list(df.ada_vector.values),
    df.Category,
    test_size=0.2
)
```

Testing the Data

```
from sklearn.ensemble import RandomForestClassifier

# Create 100 empty trees – we will use these for our classifier
clf = RandomForestClassifier(n_estimators=100)

# now fill all the trees with our training set and related category or value
#This is done randomly for us by the “fit” method
clf.fit(X_train, y_train)

# use the trees to vote on our set of test strings
y_pred = clf.predict(X_test)
```

Viewing the Results

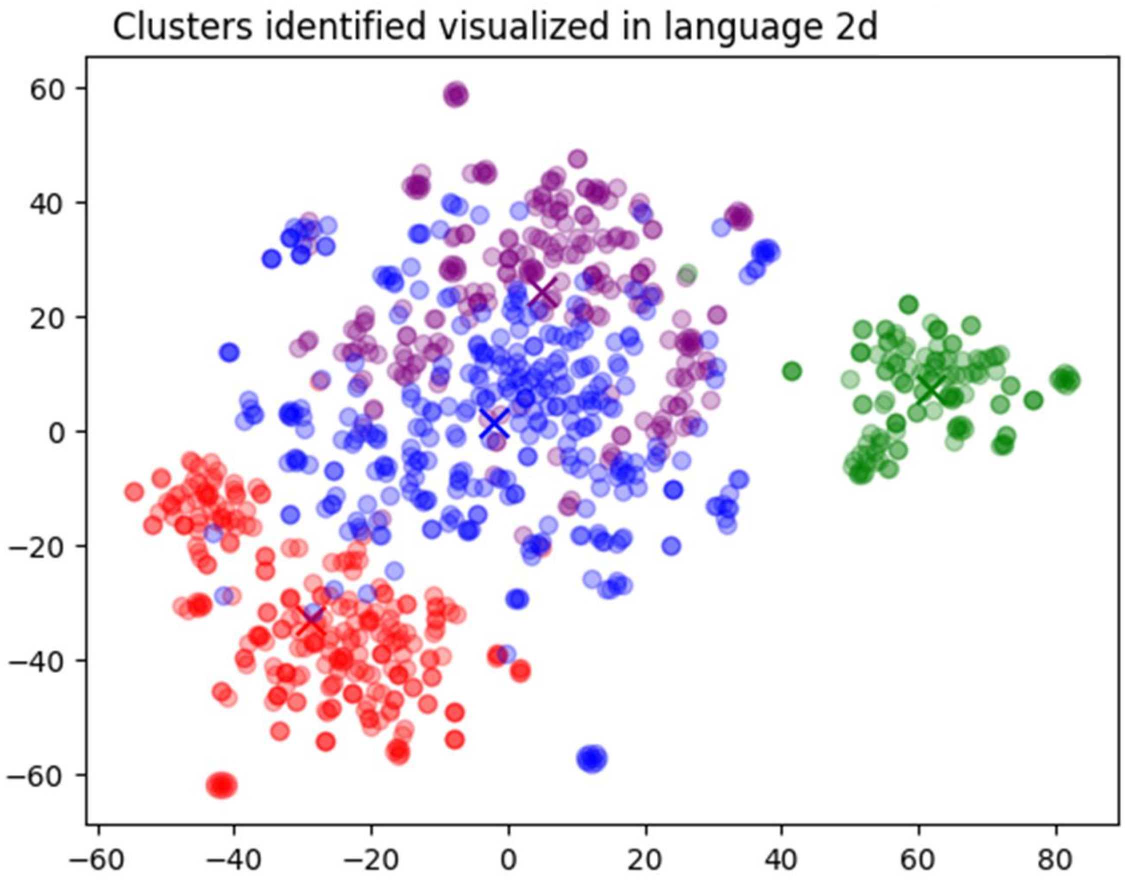
```
from sklearn.metrics import accuracy_score

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",accuracy_score(y_test, y_pred))
```

```
('Accuracy:', 0.93333333333333335)
```

Clustering

Clustering

- Find hidden groupings within a random data set
- Before Classification or Categorization established
- Unsupervised


Clusters identified visualized in language 2d
60
40
20
0
−20
−40
−60
−60
−40
−20
0
20
40
60
80

Loading the Data Set

```
import pandas as pd
import numpy as np

# read the csv file
df = pd.read_csv('output/embedded_reviews.csv')

# convert the strings stored in the ada_vector column
# into vector objects
df['ada_vector'] = df.ada_vector.apply(eval).apply(np.array)
```

Structure in Memory

- Time
- ProductID
- UserID
- Rating
- Summary
- Text
- combined
- n_tokens
- ada_vector (as a vector)

Place the Vectors into a List

```
# this is our training data
vectorlist = np.vstack(df.ada_vector.values)
```

Find the Centroids

```
from sklearn.cluster import KMeans

# setup our Kmeans parameters – haven’t done anything with data yet
results = KMeans(
                n_clusters=4,
                init="k-means++",
                random_state=42,
                n_init='auto')

# now we fit our vectorlist on top of the results model
results.fit(vectorlist)
```

Extract and Store the Clusters

```
# results has our labels (0,1,2,3)
# we end up with an array of int32 values – one for each row of data
labels = results.labels_

# Create a new column in our data frame and fill it with
# the appropriate label for each row
df["ClusterNum"] = labels
```

Structure in Memory

- Time
- ProductID
- UserID
- Rating
- Summary
- Text
- combined
- n_tokens
- ada_vector
- ClusterNum

Ask GPT What is Common

```
testcluster = 0

reviewsamples = "\n".join(
        df[df.ClusterNum == testcluster]
        .combined.str.replace("Title: ", "")
        .str.replace("\n\nContent: ", ":  ")
        .str.replace("\n", " ")
        .sample(3, random_state=42)
        .values
    )
```

Ask Davinci

```
response = openai.Completion.create(
    engine="davinci",
    prompt=f'What do the following customer reviews have in common?\n\nCustomer reviews:\n"""\n{reviews}\n"""\n\nTheme:',
    temperature=0,
    max_tokens=64
)

# What did Davinci GPT think?
print(response["choices"][0]["text"].replace("\n", ""))
```

```
All reviews mention the good flavor of the bars
```

Embedding Summary

What it is and what is isn’t

- Training Data is kept in your local database
- We cannot use this to train GPT
- Do a one-off run to calculate and store vectors
- Calculate a vector when you need it to compare to your local database

Cautions

- Doesnt use the same moderation
- Can result in social bias, sterotypes or negative sentiment toward certain groups
- English only
- Unaware of Recent events

Creative Writing and SEO

Generating a New Article

Brainstorm blog post outline ideas for the topic <x>:

Create an outline for an article about <x>:

What are 5 key points I should know when writing about <topic1>?

Write 3 engaging and informative paragraphs about "<point 1 description>":

Write 3 engaging and informative paragraphs about "<point 2 description>":

Generating a New Article

<concatenated content>

WRITE A SHORT SUPER ENGAGING BLOG POST INTRODUCTION:

<concatenated content>

WRITE A SHORT SUPER ENGAGING CONCLUSION:

Expanding Text

Make the following passage longer. Expand, expound, and elaborate on all points. Add lots of detail and new ideas.

PASSAGE:

{the original text}

REWRITTEN LONGER VERSION:

Expanding Text

Brainstorm many different ideas about how to expand the following passage:

PASSAGE:

{the original text}

DIVERGENT IDEAS TO EXPAND:

Expanding Text

Write another passage to follow the first. Explore and elaborate on the ideas listed below:

IDEAS:

{a single idea, or the list from the previous prompt}

FIRST PASSAGE:

{the original text}

NEXT PASSAGE BASED ON ABOVE IDEAS:

Summarizing

Summarize the following article:

ARTICLE:

{the original text}

SUMMARY:

Extraction

Extract the 5Ws from the following article. There should be one sentence for the who, what, when, where, and why of the article:

ARTICLE:

{the original text}

5WS:

Rewriting

Rewrite the following article in a conversational tone for an online blog:

ARTICLE:

{the original text}

REWRITE USING CONVERSATIONAL TONE:

Making a ChatBot

Without using ChatGPT

The Basics

```
response = openai.Completion.create(
        engine="davinci",
        prompt="The following is a conversation with an AI assistant. The assistant is helpful, creative, clever, and very friendly.\n\nHuman: Hello, who are you?\nAI: I am an AI created by OpenAI. How can I help you today?\n\nHuman: What is RAM?\nAI:",
        temperature=0.9,
        max_tokens=150,
        top_p=1,
        stop=["\n", " Human:", " AI:"]
)
```

The Response

```
{
 "choices": [
   {
     "finish_reason": "stop",
     "text": " RAM is a computer storage space used to temporarily store data while the computer is processing it."
   }
 ]
}
```

- `response["choices"][0]["text"].replace("\n", "")`

[not a complete response]

Continuing the Conversation

```
response = openai.Completion.create(
        engine="davinci",
        prompt="The following is a conversation with an AI assistant. The assistant is helpful, creative, clever, and very friendly.\n\nHuman: Hello, who are you?\nAI: I am an AI created by OpenAI. How can I help you today?\n\nHuman: How does RAM work?\nAI:",
        temperature=0.9,
        max_tokens=150,
        top_p=1,
        presence_penalty=0.6,
        stop=["\n", " Human:", " AI:"]
)
```

Issues

- Without training, the BOT will tend to make things up

- No company specific knowledge

Training a Service Bot

- Create a file of 200+ unique questions and the answers provided by a real human
  - These can be sourced from real-life chat logs

- Store these in prompt and content pairs
- Don't forget the leading space in the content!

- Make sure the list is curated and don't include prompts that would be considered as "chit chat"

- Set temperature close to 0 (but not zero)

Product Knowledge

- Create prompts that ask common questions about a product
  - Q: What does … do\nA:
  - Q: How do I …\nA:

- Create content that answers the question
  - Use GPT3 to summarize large blocks of text

- Create "What is product A" prompts with associated full description

Embedding + Completion

- When you have too many blocks
  - Documents, white papers etc

- Process:
  - Break text/doc into chunks (paragraphs) up to 2k
  - Store in local database
  - Use Encoding to generate embedding vectors

  - User asks a question
  - Use Embedding and semantic search to find record
  - Reformat question and send to GPT
  - May not need Davinci (Cost Saving)

GPT Reformatted Question

Answer the question as truthfully as possible using the provided text, and if the answer is not contained within the text below, say “I don't know”

Context:

<your chunk goes here>

Q: <users question goes here>

A:

- temperature = 0

Ringfencing the Chatbot

- Hard to stop users going off track
- Misuse = Cost to you

- Try Embedding
- If the dot product is too low, they user may be off track

Persistent Memory

- No memory of the chat
- Use Embedding
- Find top 4 or 4 interactions
- Include them with prompt to provide relevant context

Chatbot Recipe

- I want you to act as a …
- I will give you …
- and you will provide … including …
- You should only reply with …., and nothing else.
- Do not write explanations.
- My first request is "…

Sample Recipe

I want you to act as a tech reviewer.

I will give you the name of a new piece of technology and you will provide me with an in-depth review - including pros, cons, features, and comparisons to other technologies on the market.

You should only reply with your review, and nothing else. Do not write explanations.

My first suggestion request is "I am reviewing iPhone 11 Pro Max"

Usage Policies

safety-specialists@openai.com

Legality

- Promoting Dishonesty
- Illegal or Harmful Industries
- Misuse of Personal Data
- Deceiving or Manipulating Users
- Trying to Influence Politics

Content

- Hate
- Harassment
- Violence
- Self-harm
- Sexual
- Political
- Spam
- Deception
- Malware

Risk of Potential Harm

- Criminal Justice
- Law Enforcement
- Legal
- Government and Civil Services
- Healthcare
- Mental Health
- Therapy
- Wellness
- Coaching
- Finance

Best Safety Practices

Safety Best Practices

- Use the FREE Moderation Endpoint
- Adversarial Testing
- Human Reviews
- Rate Limits
- Good Prompt Engineering

Safety Best Practices

- Require User to Login
- Watch Out for Overflows
- Provide a User Feedback Loop
- Communicate Limitations
- Track End User IDs

API Key Safety

- Give each developer their own API key
- Never commit your key to a repository
- Use Server based environmental variables
- Rotate API Key
- NEVER include API keys in front facing code (Eg Javascript)

Prompt Injection

- Protect your prompt
- Use Regex to check your responses
- Fine tune it over time
- Google “Avoid Prompt Injection”

Bonus : Cost Saving Tips

How to save money on API calls

Use the GPT-3 Token Estimator

- Free tool
- Shows breakdown of words
- Quick way to compare cost of various phrase options

Prompt Paraphrasing

Write a creative ad for the following product to run on Facebook aimed at parents:

16 tokens

- Using QuillBot Paraphraser (quillbot.com)

Create a clever Facebook ad for the following product aimed towards parents:

13 tokens

Swap Out Names

Classify the sentiment in the following sentence as positive, negative, or neutral.

Samantha loves working in the office

8 tokens

vs

John loves working in the office

6 tokens

Multiple Tasks

Translate this into 1. French, 2. Spanish and 3. Japanese:

What rooms do you have available?

1.

Select the Best Model for the Task

- Davinci
- Curie
- Babbage
- Ada

Use Training

- Cost to Setup

- More expensive to use

- No need to use multi-shot prompts

Use Embedding and Offline Data

- Embedding is significantly cheaper
- Excellent for very large data sets

- More code required

- No need to train

- Use "Context:" in the question when you eventually call GPT