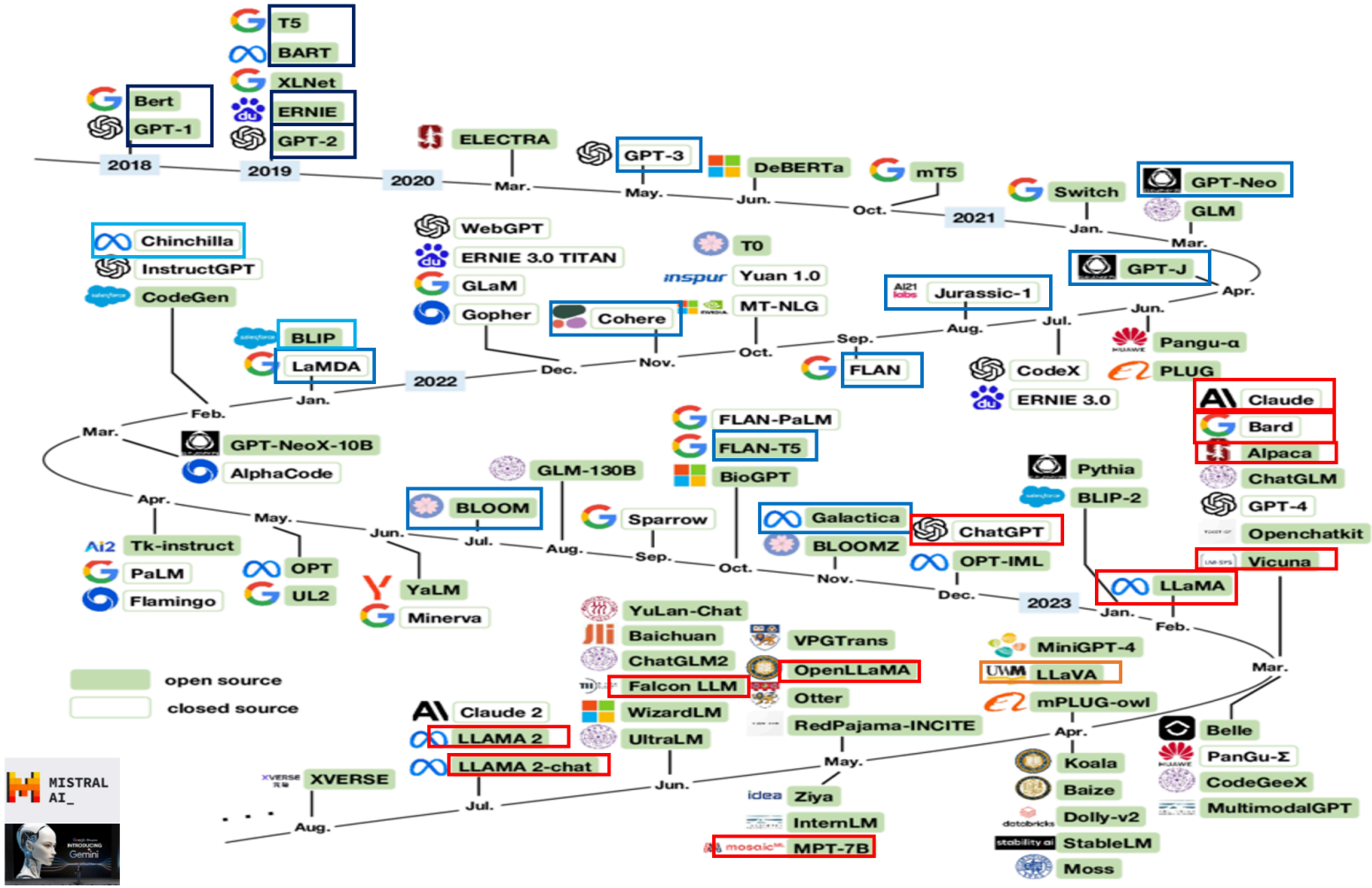
Text Generation using OpenAI LLMs (GPT 3.5/GPT4)

Access this code on Github - https://github.com/abhinav-kimothi/OpenAI-Marvels



- Installation & Importing Libraries

- Accessing OpenAI (Keys) & Setting up Client

- Chat Completion API (Basic Text Generation)
- Tokens
- Embeddings
- RAG
- Fine Tuning

- Time (75 minutes coding + 15 minutes QnA)
- We'll also take questions at the end of each section
- Aim is to explore OpenAI features

Installation & Importing Libraries

In [ ]:
```
%pip install --upgrade pip --quiet
```

```
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:
```
%pip install -r ../requirements.txt --quiet
```

```
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:
```
import openai #OpenAI python library
from openai import OpenAI #OpenAI Client
from configparser import ConfigParser #library to read the config file

import tiktoken #library to count tokens

import gradio as gr #library for gradio interface

from sklearn.metrics.pairwise import cosine_similarity #for calculating similarities between embeddings

from bs4 import BeautifulSoup #for extracting text from webpages
import requests
```

```python
import PyPDF2 #for reading text from pdf

from langchain.document_loaders import TextLoader #to load text for RAG
from langchain.text_splitter import RecursiveCharacterTextSplitter #to chunk data for RAG
from langchain.embeddings.openai import OpenAIEmbeddings #to crate embeddings for RAG
from langchain.vectorstores import FAISS # to store embeddings in a vector index
from sklearn.model_selection import train_test_split #for model finetuning

import json #for creating finetuning files

import numpy as np
from sklearn.cluster import KMeans # for clustering of text using embeddings

import pandas as pd

import warnings
warnings.filterwarnings('ignore')
```

```
/Users/kim/Desktop/Github/OpenAI-Marvels/.venv/lib/python3.11/site-packages/tqdm/auto.py:21: TqdmWarning: IProgres
s not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.ht
ml
  from .autonotebook import tqdm as notebook_tqdm
```

Accessing OpenAI (Keys) & Setting up Client

Question : Have you procured your OpenAI API key?

If not, create one here - https://platform.openai.com/api-keys

Also, check if you have sufficient balance - https://platform.openai.com/account/billing/overview

Step 1 : Read and set the OpenAI API key in the environment

In [ ]:
```
config_object = ConfigParser()
config_object.read("../config.ini")
openai.api_key = config_object["OPENAI"]["openai_key"] #read the api key from the config file
```

In [ ]:
```
#openai.api_key="<Your API Key>"
```

There are several ways of storing the API key in the environment. You may choose as per your preference or your organisation's policy

Step 2: Initialize the OpenAI client. This serves as an interface to interact with OpenAI's services and APIs.

In [ ]:
```
client = OpenAI(api_key=openai.api_key)
```

And we're ready! Let's try and make the first call!

In [ ]:
```
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "user", "content": "Hello!"}
  ]
)
print(response.choices[0].message.content)
```

```
Hello! How can I assist you today?
```

Chat Completion API (Basic Text Generation)

Introduction

>>Generative AI, and LLMs specifically, is a General Purpose Technology that is useful for a variety of applications

"LLMs can be, generally, thought of as a next word prediction model"

What is an LLM?

- LLMs are **machine learning models** that have learned from **massive datasets** of human-generated content, finding statistical patterns to replicate human-like abilities.
- **Foundation models**, also known as base models, have been trained on trillions of words for weeks or months using extensive compute power. These models have **billions of parameters**, which represent their memory and enable sophisticated tasks.
- **Interacting with LLMs differs from traditional programming paradigms. Instead of formalized code syntax, you provide natural language prompts to the models**.
- When you pass a **prompt** to the model, it predicts the next words and generates a **completion**. This process is known as **inference**.

Prompts, Completions and Inference!

Available OpenAI models

GPT 4

(Production)

Name | Context Window | Cut-off date | Snapshot

gpt-4 | 8,192 tokens | Up to Sep 2021 | gpt-4-0613

gpt-4-32k | 32,768 tokens | Up to Sep 2021 | gpt-4-32k-0613

(Preview)

gpt-4-turbo-preview | 128,000 tokens | Up to Dec 2023 | gpt-4-1106-preview

gpt-4-vision-preview | 128,000 tokens | Up to Apr 2023 | gpt-4-1106-vision-preview

---

GPT 3.5

gpt-3.5-turbo | 16,385 tokens | Up to Sep 2021 | gpt-3.5-turbo-1106

gpt-3.5-turbo-instruct | 4,096 tokens | Up to Sep 2021

For more details, visit the official documentation -> https://platform.openai.com/docs/models

---

IMP : **"model"** is passed as a parameter in the chat completions API

OpenAI Chat Messages (Prompt) Structure

Role : OpenAI allows for *three* roles/personas -

1. **System** : The overarching constraints/definitions/intructions of the system that the LLM should "remember"
2. **User** : Any instruction a user wants to pass to the LLM
3. **Assistant** : The response from the LLM

Content : Any message or "prompt" of these personas are passes as "Content"

Why is this important? : Makes it easier to adapt an LLM to store conversation history.

IMP : **"role"** and **"content"** are passed as a dictionary in the **"messages"** parameter in the chat completions API

Let's try!

In [ ]:
```
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "system", "content": "You are a helpful assistant knowledgeable in the field of Cricket."},
    {"role": "user", "content": "When did Australia win their first Cricket World Cup?"}
  ]
)
```

In [ ]:
```
print(response.choices[0].message.content)
```

```
Australia won their first Cricket World Cup in the year 1987. They defeated England in the final to claim their ma
iden title.
```

In [ ]:
```
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "system", "content": "You are a helpful assistant knowledgeable in the field of Cricket."},
    {"role": "user", "content": "When did Australia win their first Cricket World Cup?"},
    {"role": "assistant", "content": "Australia won their first Cricket World Cup in the year 1987. They defeated E
    {"role": "user", "content": "How much did they score?"}
  ]
)

from pprint import pprint

pprint(response.choices[0].message.content)
```

```
('In the final of the 1987 Cricket World Cup, Australia scored 253 runs for '
 'the loss of 5 wickets in their allotted 50 overs. England, in response, '
 'could only manage 246 runs, giving Australia a 7-run victory.')
```

Chat API Parameters

>>**"model"** and **"messages"** are the two required API parameters

There are several other optional parameters that help configure the response

---

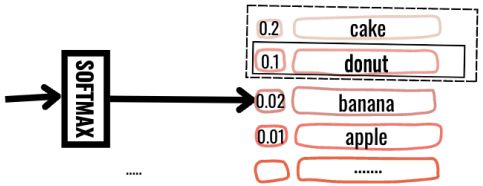
n : Number of responses you want the LLM to generate for the instriction

max_tokens : Maximum number of tokens you want to restrict the Inference to (This includes both the prompt/messages and the completion)

temperature : Temperature controls the "randomness" of the responses. Higher value increases the randomness; lower value makes the output deterministic (value between 0 and 2)

top_p : The model considers the results of the tokens with top_p probability mass (value between 0 and 1)

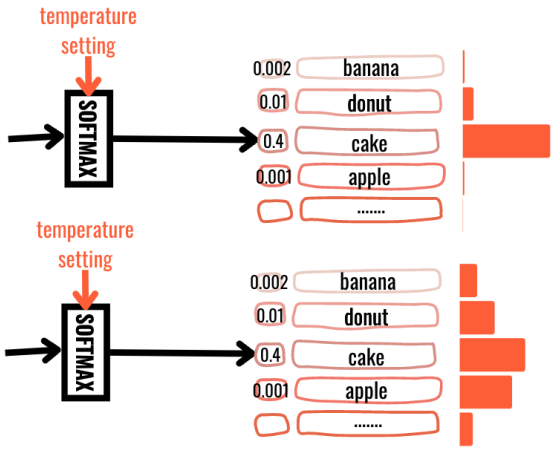
Top P.



Top P : The word/token is selected using random-weighted strategy but only from amongst the top words totalling to probability <=P

Here for P=0.33, one of cake or donut will be selected randomly but apple or banana will never be selected

Temperature



Cooler Temperature (lesser value) :
The distribution is strongly peaked

Warmer Temperature (higher value) :
Flatter probability distribution

IMP : It is recommended to configure either one of "temperature" and "top_p" but not both

frequency_penalty : Penalize new tokens based on their existing frequency in the text so far *(Value between -2 and 2)*

presence_penalty : Penalize new tokens based on whether they appear in the text so far *(Value between -2 and 2)*

logprobs : Flag to return log probability of the generated tokens *(True/False)*

logit_bias : Parameter to control the presence of particular tokens in the output *(Value between -100 and 100)

response_format : Response of the model can be requested in a particular format *(Currently : JSON and Text)*

seed : Beta feature for reproducible outputs (setting a seed value may produce the same output repeatedly)

stop : End of Sequence tokens that will stop the generation

stream : To receive partial message deltas *(True/False)*

user : ID representing end user (This helps OpenAI detect abuse. May be mandatory for higher rate limits)

tools : used in function calling

tool_choice : used in function calling

In [ ]:
```
def gpt_call(model:str="gpt-3.5-turbo",prompt:str="Have I provided any input",n:int=1,max_tokens:int=100,temperatur


    response = client.chat.completions.create(
    model=model,
    messages=[
       {"role": "user", "content": prompt}
    ],
    max_tokens=max_tokens,
    temperature=temperature,
    presence_penalty=presence_penalty,
    n=n
    )

    output=''

    for i in response.choices:
        output+=str(i.message.content)+'\n----------\n'



    return output
```

In [ ]:
```
print(gpt_call(prompt="Write a title for a workshop on openai API",n=2,temperature=0))
```

```
"Unlocking the Power of OpenAI: A Hands-On Workshop on Harnessing the OpenAI API"
------------
"Unlocking the Power of OpenAI: A Hands-On Workshop on Harnessing the OpenAI API"
------------
```

In [ ]:
```python
def text_to_uppercase(model,n,max_tokens,temperature,presence_penalty,prompt):
    return str(type(n)) + str(type(temperature))

model=gr.Radio(["gpt-4","gpt-3.5-turbo"], label="Select Model")
n=gr.Radio([1,2,3], label="Number of Responses")
max_tokens=gr.Slider(minimum=10, maximum=500, label="Maximum Tokens")
temperature=gr.Slider(minimum=0.0, maximum=1.0, label="Temperature")
prompt=gr.Text(label="Prompt")


iface = gr.Interface(fn=gpt_call, inputs=[model,prompt,n,max_tokens,temperature], outputs="text")
iface.launch()
```

In [ ]:
```python
iface.close()
```

```
Closing server running on port: 7882
```

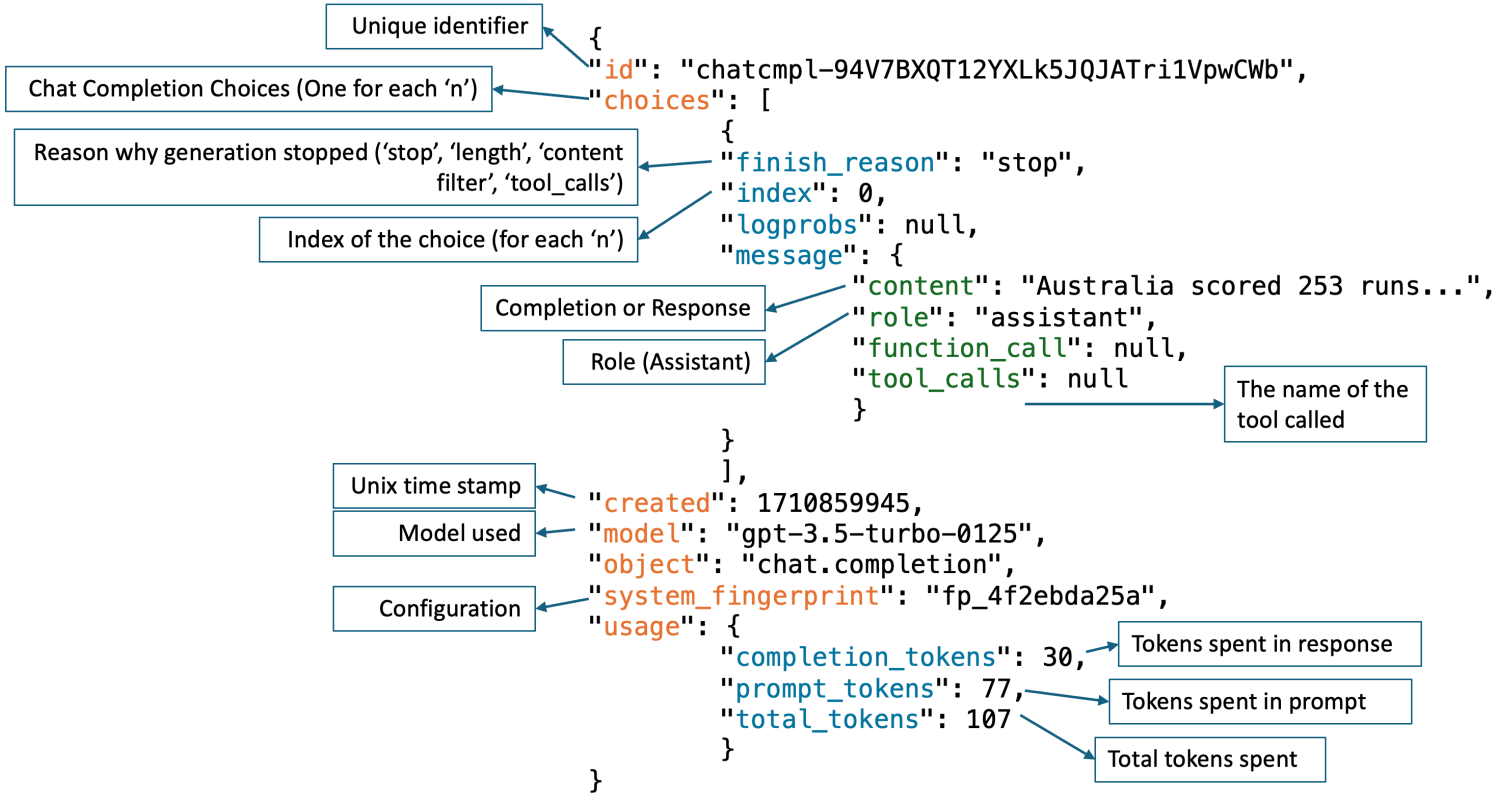
Decoding the Response Object

In [ ]:
```python
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "system", "content": "You are a helpful assistant knowledgeable in the field of Cricket."},
    {"role": "user", "content": "When did Australia win their first Cricket World Cup?"},
    {"role": "assistant", "content": "Australia won their first Cricket World Cup in the year 1987. They defeated E
    {"role": "user", "content": "How much did they score?"}
  ]
)
```

In [ ]:
```python
print((response.model_dump_json(indent=5)))
```

```
{
     "id": "chatcmpl-995GVNXDheol2CTW4BE8pHP42kL0C",
     "choices": [
          {
               "finish_reason": "stop",
               "index": 0,
               "logprobs": null,
               "message": {
                    "content": "In the 1987 Cricket World Cup final, Australia scored 253 runs for the loss of 5 w
ickets in their allotted 50 overs. England, in response, scored 246 runs, falling short by 7 runs, giving Australi
a their first Cricket World Cup victory.",
                    "role": "assistant",
                    "function_call": null,
                    "tool_calls": null
               }
          }
     ],
     "created": 1711952219,
     "model": "gpt-3.5-turbo-0125",
     "object": "chat.completion",
     "system_fingerprint": "fp_3bc1b5746c",
     "usage": {
          "completion_tokens": 56,
          "prompt_tokens": 77,
          "total_tokens": 133
     }
}
```

Chat Completion Object



Streaming

In [ ]:
```
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
```

```
  messages=[
    {"role": "system", "content": "You are a helpful assistant knowledgeable in the field of Cricket."},
    {"role": "user", "content": "When did Australia win their first Cricket World Cup?"},
    {"role": "assistant", "content": "Australia won their first Cricket World Cup in the year 1987. They defeated E
    {"role": "user", "content": "How much did they score?"}
  ],
  stream=True
)

for chunk in response:
  print(chunk.choices[0].delta.content)
```

In
 the

198
7
 Cricket
 World
 Cup
 final
,
 Australia
 scored

253
 runs
 for
 the
 loss
 of

5
 w
ickets
 in
 their
 allotted

50
 overs
.
 England
,
 in
 reply
,
 could
 only
 manage

246
 runs

```
,
 falling
 short
 by

7
 runs
,
 giving
 Australia
 their
 first
 World
 Cup
 title
.
None
```

In [ ]:
```python
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "user", "content": "Write an email requesting for a leave of absence"}
  ],
  stream=True
)

for chunk in response:
  print(chunk.choices[0].delta.content)
```

JSON

In [ ]:
```python
prompt="generate the entire text for a blog on a cricket match. \"Title\" is a catchy and attractive title of the b
```

In [ ]:
```python
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
        {"role": "user", "content": prompt}
  ],
  response_format={ "type": "json_object" }
```

```
)

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

```
{
  "Title": "Thrilling Cricket Match: A Nail-biting Encounter Between Rivals",
  "Heading1": "The Buildup",
  "Body1": "The anticipation was palpable as the date of the match approached. Both teams were in top form and fan
s were eagerly waiting for the clash between the two fierce rivals.",
  "Heading2": "The Toss",
  "Body2": "The toss was won by Team A, who elected to bat first. It was a crucial decision as the pitch was expec
ted to deteriorate later in the day.",
  "Heading3": "Team A's Innings",
  "Body3": "Team A got off to a solid start, with their top order batsmen scoring freely. However, they suffered a
collapse in the middle overs, with Team B's bowlers wreaking havoc. They managed to post a competitive total of 25
0 runs.",
  "Heading4": "Team B's Chase",
  "Body4": "Team B's chase got off to a shaky start, losing early wickets. However, their middle order batsmen ste
adied the ship and kept them in the game. The match went down to the wire, with Team B needing 10 runs off the las
t over.",
  "Heading5": "The Thrilling Finish",
  "Body5": "In a nail-biting finish, Team B's lower order batsman hit a boundary off the last ball to win the matc
h for their team. The crowd erupted in joy as Team B celebrated their hard-fought victory.",
  "Conclusion": "It was a thrilling match that had fans on the edge of their seats throughout. Both teams put up a
great fight, but it was Team B who emerged victorious in the end. Cricket fans will surely remember this match for
a long time to come."
}
```

Tokens

>>Tokens are the fundamental units of NLP

These units are typically words, punctuation marks, or other meaningful substrings that make up the text

Counting the number of tokens becomes important because -

- Number of Tokens determine the amount of computation required and hence the cost you incur
- Context Window or the maximum number of tokens an LLM can process in one go is limited

Counting Tokens

In [ ]:
```
####num_tokens_from_string function to count number of tokens in a text string
####uses tiktoken to count number of tokens in a text string
####parameters: "string" is the text string, "encoding_name" is the encoding name to be used by tiktoken
####returns: num_tokens->number of tokens in the text string
####This function is used within extract_data, extract_page, extract_YT, extract_audio, extract_image functions
def num_tokens_from_string(string: str, encoding_name="cl100k_base") -> int: #### Function to count number of token
    encoding = tiktoken.get_encoding(encoding_name) #### Initialize encoding ####
    return len(encoding.encode(string)) #### Return number of tokens in the text string ####
```

In [ ]:
```
num_tokens_from_string("Hello how are you?")
```

Out[ ]:
```
5
```

In [ ]:
```
with open("../Assets/Data/alice_in_wonderland.txt") as f:
    AliceInWonderland = f.read()

num_tokens_from_string(AliceInWonderland)
```

Out[ ]:
```
38680
```

Pricing

gpt-3.5-turbo-0125 | PROMPT - $0.50/1Mtokens|RESPONSE-$1.50 / 1M tokens

gpt-4 | PROMPT - $30.00/1Mtokens|RESPONSE-$60.00 / 1M tokens

gpt-4-turbo | PROMPT - $10.00/1Mtokens|RESPONSE-$30.00 / 1M tokens

---

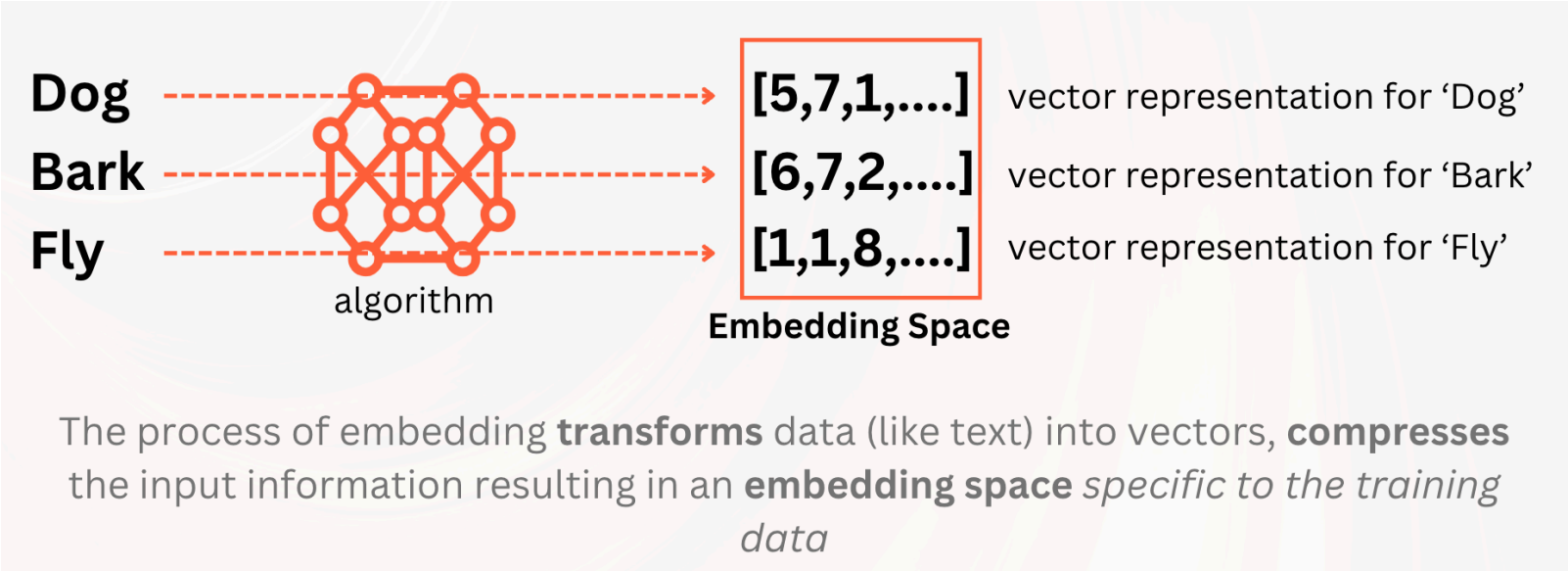
Embeddings

Introduction

>>Embeddings are vector representations of data that capture meaningful relationships between entities

These units are typically words, punctuation marks, or other meaningful substrings that make up the text

- All Machine Learning/AI models work with numerical data. Before the performance of any operation all text/image/audio/video data has to be transformed into a numerical representation
- As a general definition, embeddings are data that has been transformed into n-dimensional matrices for use in deep learning computations.



OpenAI Embeddings

text-embedding-3-small | $0.02 / 1M tokens

text-embedding-3-large | $0.13 / 1M tokens

ada v2 | $0.10 / 1M tokens

In [ ]:
```
embeddings=client.embeddings.create(
  model="text-embedding-3-small",
  input="The food was delicious",
  encoding_format="float",
)
```

In [ ]:
```
print(embeddings.model_dump_json(indent=4))
```

In [ ]:
```
embeddings=client.embeddings.create(
  model="text-embedding-3-small",
  input=["The food was delicious","The ambience was nice","The service was ordinary"],
  encoding_format="float",
  dimensions=10
)
```

In [ ]:
```
print(embeddings.model_dump_json(indent=4))
```

```
{
    "data": [
        {
            "embedding": [
                -0.16478635,
                -0.18134578,
                -0.5129379,
                -0.32290855,
                0.0938535,
                -0.2699992,
                -0.0649755,
                0.5856378,
                -0.073911525,
                -0.37177902
            ],
            "index": 0,
            "object": "embedding"
        },
        {
            "embedding": [
                -0.029102806,
                -0.34336767,
                -0.66657615,
                -0.37806797,
                0.08133914,
                -0.4815079,
                -0.12963039,
                0.060725518,
                -0.060477655,
                -0.17713675
            ],
            "index": 1,
            "object": "embedding"
        },
        {
            "embedding": [
                -0.27565208,
                0.1992017,
                -0.5211547,
                -0.37973946,
                -0.046884183,
```

```
                -0.20189361,
                -0.28731704,
                0.56458426,
                0.01179956,
                0.15532349
            ],
            "index": 2,
            "object": "embedding"
        }
    ],
    "model": "text-embedding-3-small",
    "object": "list",
    "usage": {
        "prompt_tokens": 13,
        "total_tokens": 13
    }
}
```


Text Search

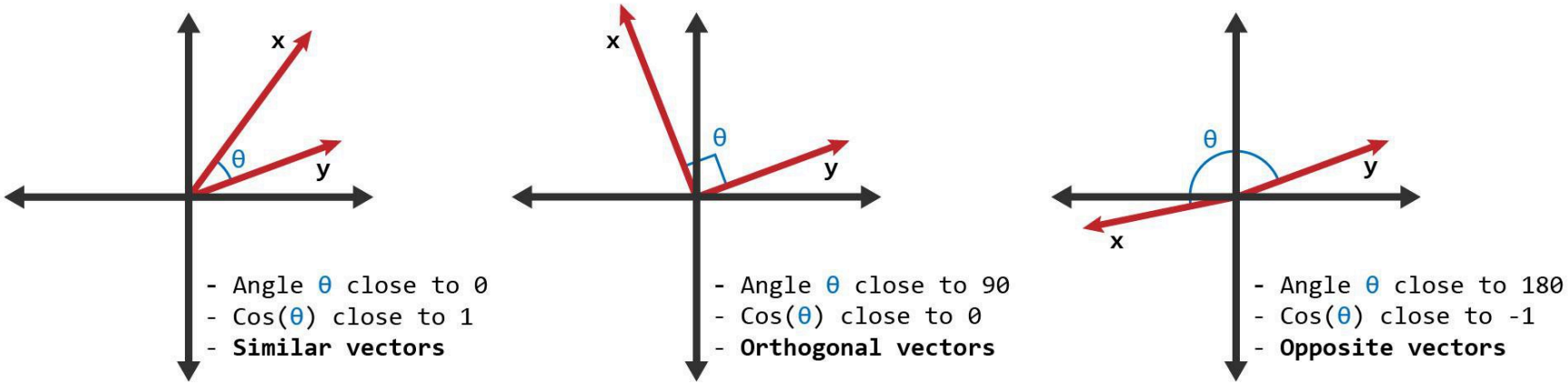
In [ ]:
```
embeddings_q=client.embeddings.create(
  model="text-embedding-3-small",
  input="food",
  encoding_format="float",
  dimensions=10
)
```

In [ ]:
```
query=embeddings_q.data[0].embedding
```

In [ ]:
```
query
```

Out[ ]:
```
[-0.012315562,
 -0.16567738,
 -0.069231726,
 -0.10728083,
 0.4480219,
 -0.61836094,
 0.29532152,
 0.4434862,
 -0.24505134,
 -0.17046502]
```

In [ ]:
```
d1=embeddings.data[0].embedding #"The food was delicious"
d2=embeddings.data[1].embedding #"The ambience was nice"
d3=embeddings.data[2].embedding #"The service was ordinary"
```



In [ ]:
```
cosine_similarity([query],[d1])
```

Out[ ]:
```
array([[0.6332542]])
```

In [ ]:
```
cosine_similarity([query],[d2])
```

Out[ ]:
```
array([[0.51180575]])
```

In [ ]:
```
cosine_similarity([query],[d3])
```

Out[ ]:
```
array([[0.28721449]])
```

1300+ Towards DataScience Medium Articles Dataset

Data Source - https://www.kaggle.com/datasets/meruvulikith/1300-towards-datascience-medium-articles-dataset

In [ ]:
```
data=pd.read_csv("../Assets/Data/medium.csv")
```

In [ ]:
```
data.head()
```

Out[ ]:

	Title	Text
0	A Beginner's Guide to Word Embedding with Gens...	1. Introduction of Word2vec\n\nWord2vec is one...
1	Hands-on Graph Neural Networks with PyTorch & ...	In my last article, I introduced the concept o...
2	How to Use ggplot2 in Python	Introduction\n\nThanks to its strict implement...
3	Databricks: How to Save Data Frames as CSV Fil...	Photo credit to Mika Baumeister from Unsplash\...
4	A Step-by-Step Implementation of Gradient Desc...	A Step-by-Step Implementation of Gradient Desc...

In [ ]:
```
data.shape
```

Out[ ]:
```
(1391, 2)
```

In [ ]:
```
trunc_data=data.iloc[0:100,:]
```

In [ ]:
```
trunc_data.shape
```

Out[ ]:
```
(100, 2)
```

In [ ]:
```
def get_embedding(text, model="text-embedding-3-small"):
   text = text.replace("\n", " ")
   return client.embeddings.create(input = [text], model=model).data[0].embedding
```

In [ ]:
```
trunc_data['embedding'] = trunc_data.Title.apply(lambda x: get_embedding(x, model='text-embedding-3-small'))
```

In [ ]:
```
trunc_data
```

Out[ ]:

	Title	Text	embedding
0	A Beginner's Guide to Word Embedding with Gens...	1. Introduction of Word2vec\n\nWord2vec is one...	[-0.024416513741016388, 0.019436374306678772, ...
1	Hands-on Graph Neural Networks with PyTorch & ...	In my last article, I introduced the concept o...	[-0.011570210568606853, -0.029422052204608917,...
2	How to Use ggplot2 in Python	Introduction\n\nThanks to its strict implement...	[-0.0054690418764948845, -0.024078190326690674...
3	Databricks: How to Save Data Frames as CSV Fil...	Photo credit to Mika Baumeister from Unsplash\...	[-0.003941336181014776, -0.022551342844963074,...
4	A Step-by-Step Implementation of Gradient Desc...	A Step-by-Step Implementation of Gradient Desc...	[-0.0020211779046803713, -0.000974284252151846...
...	...	...	...
95	Data Scientist's toolkit — How to gather data ...	Data Scientist's toolkit — How to gather data ...	[-0.026935778558254242, -0.02546488121151924, ...
96	Deep Learning on a Budget	Introduction\n\nWhy?\n\nThere are many article...	[-0.016848241910338402, -0.045139651745557785,...
97	Generating Startup names with Markov Chains	Generating Startup names with Markov Chains\n\...	[0.013092967681586742, -0.0023525876458734274,...
98	A Recipe for using Open Source Machine Learnin...	A Recipe for using Open Source Machine Learnin...	[-0.017321426421403885, -0.021059678867459297,...
99	How to Choose Between Multiple Models	How to Choose Between Multiple Models\n\nIn a ...	[0.004752688109874725, -0.0009391147177666426,...

100 rows × 3 columns

In [ ]:
```
search_string="Deep Learning"
```

In [ ]:
```
search_embedding=get_embedding(search_string)
```

In [ ]:
```
trunc_data['relevance'] = trunc_data.embedding.apply(lambda x: float(cosine_similarity([search_embedding],[x])))
```

In [ ]:
```
trunc_data
```

Out[ ]:

	Title	Text	embedding	relevance
0	A Beginner's Guide to Word Embedding with Gens...	1. Introduction of Word2vec\n\nWord2vec is one...	[-0.024416513741016388, 0.019436374306678772, ...	0.300155
1	Hands-on Graph Neural Networks with PyTorch & ...	In my last article, I introduced the concept o...	[-0.011570210568606853, -0.029422052204608917,...	0.344597
2	How to Use ggplot2 in Python	Introduction\n\nThanks to its strict implement...	[-0.0054690418764948845, -0.024078190326690674...	0.109014
3	Databricks: How to Save Data Frames as CSV Fil...	Photo credit to Mika Baumeister from Unsplash\...	[-0.003941336181014776, -0.022551342844963074,...	0.169241
4	A Step-by-Step Implementation of Gradient Desc...	A Step-by-Step Implementation of Gradient Desc...	[-0.0020211779046803713, -0.000974284252151846...	0.388754
...	...	...	...	...
95	Data Scientist's toolkit — How to gather data ...	Data Scientist's toolkit — How to gather data ...	[-0.026935778558254242, -0.02546488121151924, ...	0.313273
96	Deep Learning on a Budget	Introduction\n\nWhy?\n\nThere are many article...	[-0.016848241910338402, -0.045139651745557785,...	0.719206
97	Generating Startup names with Markov Chains	Generating Startup names with Markov Chains\n\...	[0.013092967681586742, -0.0023525876458734274,...	0.165855
98	A Recipe for using Open Source Machine Learnin...	A Recipe for using Open Source Machine Learnin...	[-0.017321426421403885, -0.021059678867459297,...	0.387291
99	How to Choose Between Multiple Models	How to Choose Between Multiple Models\n\nIn a ...	[0.004752688109874725, -0.0009391147177666426,...	0.233198

100 rows × 4 columns

In [ ]:
```
trunc_data.sort_values(by="relevance",ascending=False).iloc[0:10,:]
```

Out[ ]:

	Title	Text	embedding	relevance
96	Deep Learning on a Budget	Introduction\n\nWhy?\n\nThere are many article...	[-0.016848241910338402, -0.045139651745557785,...	0.719206
79	Applied AI: Going From Concept to ML Components	Opening your mind to different ways of applyin...	[-0.016721589490771294, -0.026498831808567047,...	0.482872
73	Transfer Learning Intuition for Text Classific...	Transfer Learning Intuition for Text Classific...	[-0.022648293524980545, 0.0017097401432693005,...	0.478084
54	Reinforcement Learning Introduction	Reinforcement Learning Introduction\n\nAn intr...	[0.009179973974823952, -0.05973218381404877, 0...	0.470448
80	Wild Wide AI: responsible data science	Wild Wide AI: responsible data science\n\nData...	[0.041022028774023056, -0.00013012583076488227...	0.445177
26	Why Machine Learning Models Degrade In Production	After several failed ML projects due to unexpe...	[0.012906364165246487, 0.030608268454670906, 0...	0.437601
29	An Introduction to Recurrent Neural Networks f...	An Introduction to Recurrent Neural Networks f...	[-0.01791239343583584, -0.02631079964339733, 0...	0.424425
9	What if AI model understanding were easy?	Irreverent Demystifiers\n\nWhat if AI model un...	[-0.011677316389977932, -0.0018296980997547507...	0.422850
68	Getting Started with Google BigQuery's Machine...	While still in Beta, BigQuery ML has been avai...	[-0.034346841275691986, 0.012737160548567772, ...	0.416386
69	Review: DeepPose — Cascade of CNN (Human Pose ...	Review: DeepPose — Cascade of CNN (Human Pose ...	[0.01773509941995144, -0.03974172845482826, 0....	0.412199

Clustering

In [ ]:
```
matrix = np.vstack(trunc_data.embedding.values)
n_clusters = 4

kmeans = KMeans(n_clusters = n_clusters, init='k-means++', random_state=42)
kmeans.fit(matrix)
trunc_data['Cluster'] = kmeans.labels_
```

In [ ]:
```
trunc_data
```

Out[ ]:

	Title	Text	embedding	relevance	Cluster
0	A Beginner's Guide to Word Embedding with Gens...	1. Introduction of Word2vec\n\nWord2vec is one...	[-0.024416513741016388, 0.019436374306678772, ...	0.300155	3
1	Hands-on Graph Neural Networks with PyTorch & ...	In my last article, I introduced the concept o...	[-0.011570210568606853, -0.029422052204608917,...	0.344597	3
2	How to Use ggplot2 in Python	Introduction\n\nThanks to its strict implement...	[-0.0054690418764948845, -0.024078190326690674...	0.109014	1
3	Databricks: How to Save Data Frames as CSV Fil...	Photo credit to Mika Baumeister from Unsplash\...	[-0.003941336181014776, -0.022551342844963074,...	0.169241	2
4	A Step-by-Step Implementation of Gradient Desc...	A Step-by-Step Implementation of Gradient Desc...	[-0.0020211779046803713, -0.000974284252151846...	0.388754	3
...	...	...	...	...	...
95	Data Scientist's toolkit — How to gather data ...	Data Scientist's toolkit — How to gather data ...	[-0.026935778558254242, -0.02546488121151924, ...	0.313273	2
96	Deep Learning on a Budget	Introduction\n\nWhy?\n\nThere are many article...	[-0.016848241910338402, -0.045139651745557785,...	0.719206	3
97	Generating Startup names with Markov Chains	Generating Startup names with Markov Chains\n\...	[0.013092967681586742, -0.0023525876458734274,...	0.165855	3
98	A Recipe for using Open Source Machine Learnin...	A Recipe for using Open Source Machine Learnin...	[-0.017321426421403885, -0.021059678867459297,...	0.387291	0
99	How to Choose Between Multiple Models	How to Choose Between Multiple Models\n\nIn a ...	[0.004752688109874725, -0.0009391147177666426,...	0.233198	0

100 rows × 5 columns

In [ ]:
```
trunc_data["Prompt"] = trunc_data["Title"] + " belongs to Cluster number " + trunc_data["Cluster"].astype(str)
```

In [ ]:
```
info=''
for i in trunc_data["Prompt"]:
    info+=i+'\n'
```

In [ ]:
```
print(info)
```

In [ ]:
```
prompt="""Below is information of blog titles grouped into clusters. There are four clusters. Come up with Names fo
```

In [ ]:
```
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "user", "content": prompt}
  ]
)
```

In [ ]:
```
print(response.choices[0].message.content)
```

```
Cluster Names:
1. Data Visualization and Analysis
2. Data Management and Processing
3. Machine Learning Implementations
4. AI Ethics and Understanding
```

RAG

Introduction

>>Users look at LLMs for knowledge and wisdom, yet LLMs are sophisticated predictors of what word comes next

Hallucinations and Restricted "Parameteric" Memory are the biggest drawbacks of LLMs

- Hallucinations - Very confidently, provide incorrect information.
- Missing Knowledge - Not having information (information available only that is available the training data)

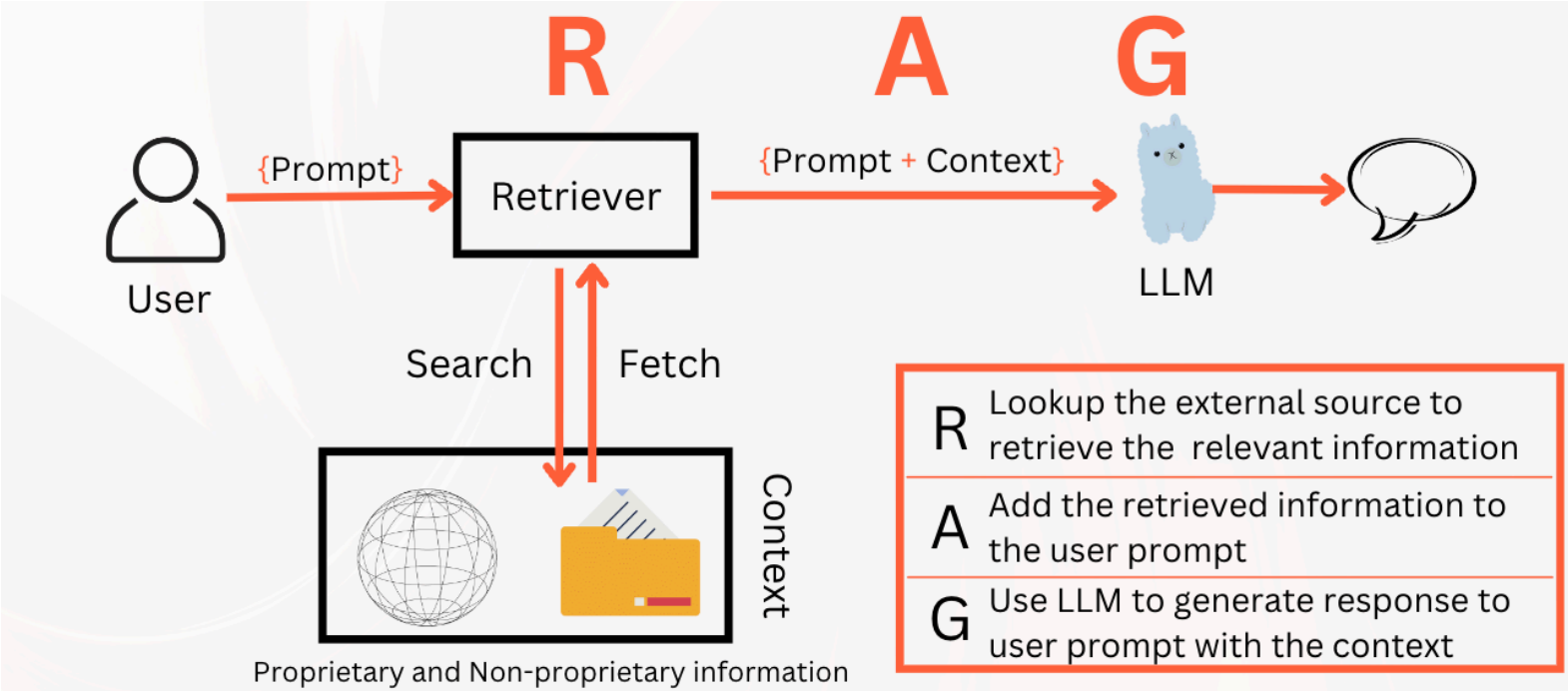
In [ ]:
```
prompt="What is amica developed by Portable?"
```

In [ ]:
```
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "user", "content": prompt}
  ]
)
```

In [ ]:
```
pprint(response.choices[0].message.content)
```

```
('Amica is a comprehensive digital solution developed by Portable, a design '
 'and technology company based in London. It is designed to provide support '
 'and information to individuals seeking asylum in the UK, helping them '
 'navigate the complex immigration system and access the services and '
 'resources they need. Amica aims to make the asylum process more transparent '
 'and accessible for refugees, empowering them to make informed decisions '
 'about their future.')
```

Retrieval Augmented Generation or RAG seems to solve these problems

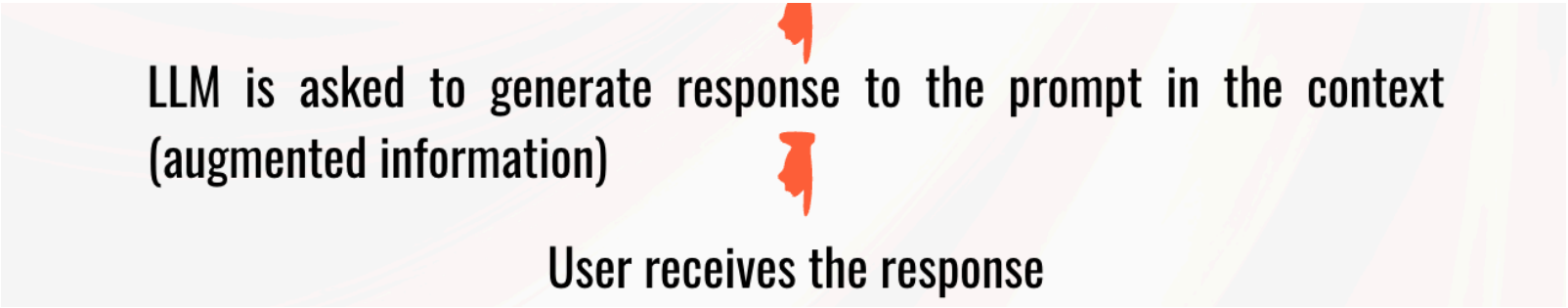


What is RAG?

User enters a prompt/query

Retriever searches and fetches information relevant to the prompt (e.g. from the internet or internet data warehouse)

Retrieved relevant information is augmented to the prompt as context



Retrieval

In [ ]:
```
def extract_page(link): #### Function to extract text from weblink ####
    address=link #### Store weblink in address variable ####
    response=requests.get(address) #### Get response from weblink using requests ####
    soup = BeautifulSoup(response.content, 'html.parser') #### Parse response using BeautifulSoup ####
    text=soup.get_text() #### Extract text from parsed response ####
    lines = filter(lambda x: x.strip(), text.splitlines()) #### Filter out empty lines ####
    website_text = "\n".join(lines) #### Join lines to form text ####
    return website_text
```

In [ ]:
```
text=extract_page("https://portable.com.au/work/amica")
```

In [ ]:
```
text
```

Augmentation

In [ ]:
```
augmented_prompt=f"You have been provided a context about below. Based only on the context answer the following que
```

Generation

In [ ]:
```
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "user", "content": augmented_prompt}
```

```
  ]
)
```

In [ ]:
```
pprint(response.choices[0].message.content)
```

```
('Amica is a digital solution developed by Portable for separating couples to '
 'help guide former partners towards an amicable resolution of family law '
 'issues.')
```

In [ ]:
```
encoding_name="cl100k_base"
encoding = tiktoken.get_encoding(encoding_name)
len(encoding.encode(text))
```

Out[ ]:
```
2083
```

In [ ]:
```
reader = PyPDF2.PdfReader("../Assets/Data/InnovatorsDilemma.pdf")
pages = reader.pages
# get all pages data
text = "".join([page.extract_text() for page in pages])
```

In [ ]:
```
len(encoding.encode(text))
```

Out[ ]:
```
117932
```

RAG At Scale

In [ ]:
```
def split_text_and_create_embeddings(text): #### Function to create embeddings from text ####
    with open('../Assets/Data/temp.txt','w') as f: #### Write text to a temporary file ####
         f.write(text) #### Write text to a temporary file ####
         f.close() #### Close temporary file ####
    loader=TextLoader('../Assets/Data/temp.txt') #### Load temporary file using TextLoader ####
    document=loader.load() #### Extract text from temporary file ####
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=10000, chunk_overlap=2000) #### Initialize text split
    docs = text_splitter.split_documents(document) #### Split document into chunks of 10000 tokens ####
    num_emb=len(docs) #### Count number of embeddings ####
    embeddings = OpenAIEmbeddings(openai_api_key=openai.api_key) #### Initialize embeddings ####
```

```
    db = FAISS.from_documents(docs, embeddings) #### Create embeddings from text ####
    return db, num_emb #### Return database with embeddings and number of embeddings ####
```

In [ ]:
```
vector_db, num_emb=split_text_and_create_embeddings(text)
```

In [ ]:
```
print(num_emb)
```

```
68
```

In [ ]:
```
vector_db.save_local(folder_path="../Assets/Data")
```

In [ ]:
```
embeddings = OpenAIEmbeddings(openai_api_key=openai.api_key)
```

In [ ]:
```
local_vectors=FAISS.load_local(folder_path="../Assets/Data/",embeddings=embeddings,allow_dangerous_deserialization=
```

In [ ]:
```
def search_context(db,query): ###### search_context function
     defin=db.similarity_search(query) ###### call the FAISS similarity_search function that searches the database
     return defin[0].page_content ###### return the most relevant section to the user question
```

In [ ]:
```
prompt="Why do great firms fail?"
```

In [ ]:
```
context=search_context(local_vectors,prompt)
```

In [ ]:
```
context
```

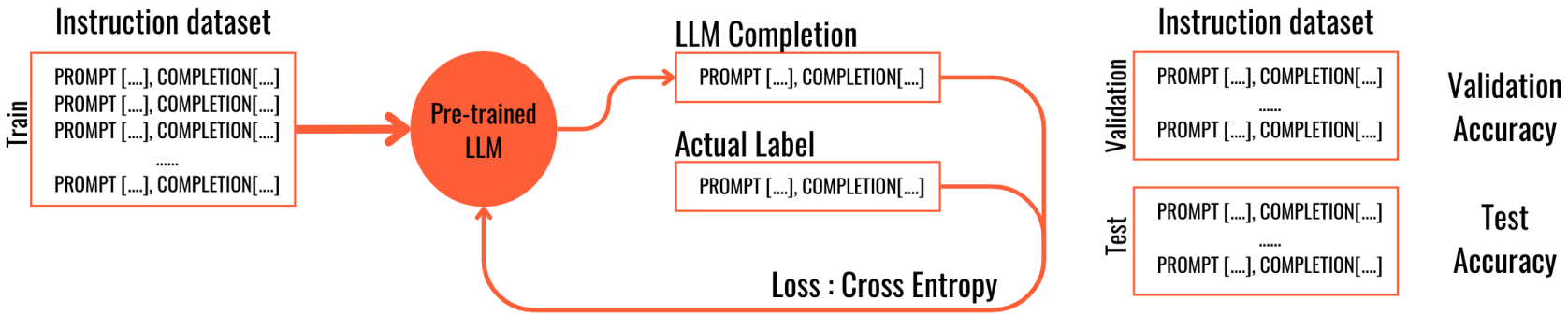
In [ ]:
```
augmented_prompt=f"You have been provided a context in [[[]]] and a prompt below. Respond to the prompt only from t
```

In [ ]:
```
pprint(client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "user", "content": augmented_prompt}
  ]
).choices[0].message.content)
```

```
('Great firms fail due to their inability to stay atop their industries when '
 'they confront certain types of market and technological change. It is not '
 'about the failure of simply any company, but of well-managed companies that '
 'have their competitive antennae up, listen astutely to their customers, '
 'invest aggressively in new technologies, and yet still lose market '
 'dominance. This failure can happen in industries that move fast or slow, '
 'built on different types of technology, and in both manufacturing and '
 'service industries.')
```

Fine Tuning

Introduction



Fine tuning process is a classification model training

Taking a general purpose model and train it to perform a specialized/specific task

- Hallucinations
- RAG Misses
- Learn New Information (When data size is large)
- Cost Optimization

- Privacy

Challenges

- Need quality data
- Upfront cost
- Expertise

There are three broad steps in LLM finetuning -

1. Data Preparation (for the specific task/use case)
2. Training (chosen training methodology)
3. Evaluation

Data Preparation

In [ ]:
```
{"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user",
{"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user",
{"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user",
```

In [ ]:
```
data_folder_path='../Assets/Data/'
training_data_file='fine_tuning_data.csv'
```

In [ ]:
```
data_for_finetuning=pd.read_csv(data_folder_path+training_data_file)
```

In [ ]:
```
data_for_finetuning.head()
```

In [ ]:
```
data_for_finetuning.shape
```

In [ ]:
```
train_data, test_data = train_test_split(data_for_finetuning, test_size=0.2, random_state=42)
```

In [ ]:
```
train_data.shape
```

In [ ]:
```
test_data.shape
```

In [ ]:
```
prompt_end="<--"
completion_end="-->"
```

In [ ]:
```
def convert_data_for_turbo(data,file):
  with open(file,'w') as f:
    for _,rows in data.iterrows():
      prompt=rows['prompt']+prompt_end
      completion=rows['completion']+completion_end
      json_line = {'messages': [{'role': 'system',
                                 'content': ''},
                                {'role': 'user',
                                 'content': prompt},
                                {'role': 'assistant',
                                'content': completion}]}
      f.write(json.dumps(json_line) + '\n')

  print(f'JSONlines file "{file}" has been created.')
```

In [ ]:
```
train_file_turbo_name='train_turbo.jsonl'
test_file_turbo_name='test_turbo.jsonl'
```

In [ ]:
```
convert_data_for_turbo(train_data,data_folder_path+train_file_turbo_name)
convert_data_for_turbo(test_data,data_folder_path+test_file_turbo_name)
```

In [ ]:
```
client.files.create(
                file=open(data_folder_path+train_file_turbo_name, "rb"),
                purpose='fine-tune'
)
```

In [ ]:
```
client.files.create(
                file=open(data_folder_path+test_file_turbo_name, "rb"),
                purpose='fine-tune'
)
```

Training

In [ ]:
```
client.fine_tuning.jobs.create(
              training_file="file-6AUxqwaqG16AFy1PVLEGFKZI",
              validation_file="file-7GByyQB3nnXSKZtgLQt542UU",
              model="gpt-3.5-turbo-0125",
              suffix="Workshop",
              hyperparameters={"n_epochs":1,
                               "batch_size":1,
                               }
              )
```

In [ ]:
```
print(client.fine_tuning.jobs.list(limit=2).model_dump_json(indent=5))
```

In [ ]:
```
print(client.fine_tuning.jobs.list_events(fine_tuning_job_id="ftjob-OP2LgxuFTe4UxHoxYbYzG7Vw", limit=2).model_dump_
```

Evaluation

In [ ]:
```
test_data.iloc[10]["prompt"]
```

In [ ]:
```
prompt=str(test_data.iloc[10]["prompt"])+prompt_end
```

In [ ]:
```
response = client.chat.completions.create(
  model="ft:gpt-3.5-turbo-0125:artificial-kimtelligence:workshop:94dD2dNr",
  messages=[
            {'role': 'system','content': ''},
            {"role": "user", "content": prompt}
  ]
)
```

In [ ]:
```
print(response.choices[0].message.content[:-3])
```

```
Homomorphic encryption may sound complex, but it enables operations on encrypted data without decryption, essentia
l in secure computati
```

In [ ]:
```
response = client.chat.completions.create(
  model="ft:gpt-3.5-turbo-0125:artificial-kimtelligence:workshop:94dD2dNr",
  messages=[
    {'role': 'system','content': ''},
    {"role": "user", "content": "explain How Homomorphic Encryption Works"}
```

```
  ]
)

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

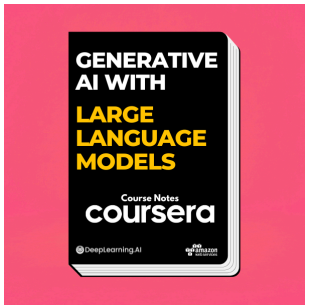
```
Homomorphic encryption may sound complex, but it enables operations on encrypted data without decryption, essentia
l in secure computation.
```

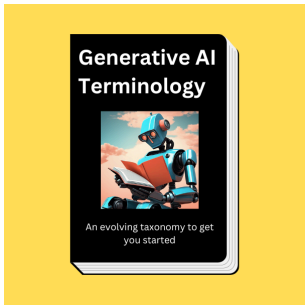
Hi! I'm Abhinav!

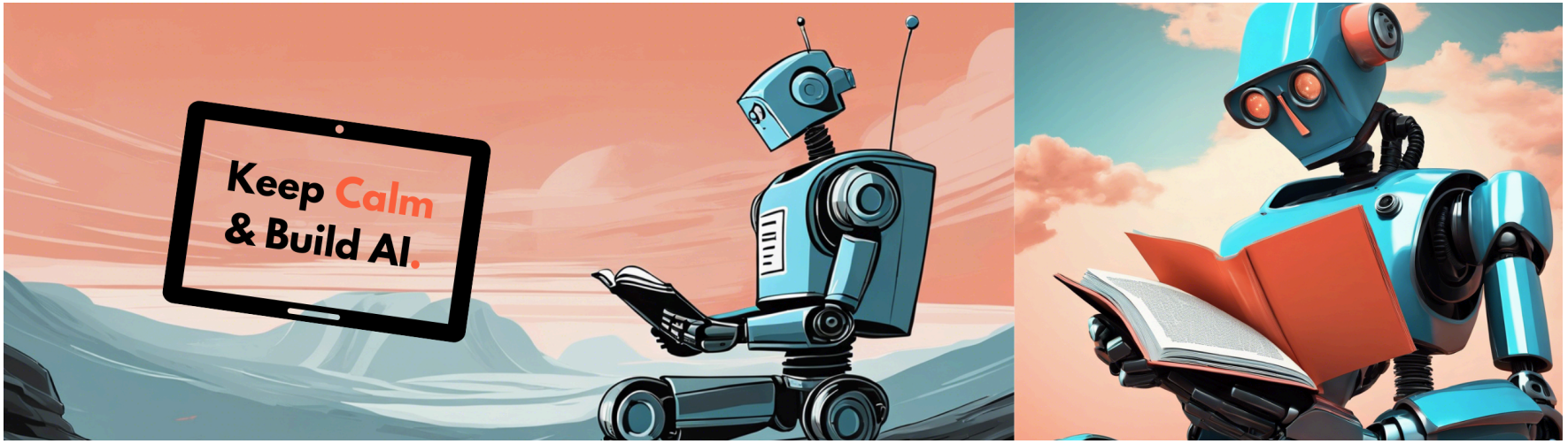
>>Hope to stay connected!

Follow 10 Medium LinkedIn eMail Follow @@

>>Also, read these for more details on Generative AI!






Keep Calm
& Build AI.