Prompt Engineering and Probing with GPT3

With GPT3, we can do a variety of tasks without the need of training a model. All we need to do is convert the task into an text generation task that follows a set of instructions called *prompts*. As an example, the task of sentiment classification can be designed as:

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

Tweet: I loved the new Batman movie!
Sentiment:
```

The GPT3 model then completes the text above with the response **Positive**. The above prompt is an example of zero-shot learning, meaning, we are not providing any signal/direction that can guide the decision and merely rely on GPT's pretraining objective:

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

Tweet: I really liked the Spiderman movie!
Sentiment: Positive

Tweet: I loved the new Batman movie!
Sentiment:
```

Now this is an example of 1-shot learning, i.e., you are providing an labeled example of how the output should look and then ask GPT to complete the next example. When you use more than 1 labeled example, it is known as few-shot learning. Generally, if you provide more examples in the prompt, it will make better predictions.

Getting Started

In this assignment, we will first need to register for an account at: https://platform.openai.com/ As a free trial, you will get $18 credits to make api calls to the GPT server. Once registered, you should go through the docs here: https://platform.openai.com/docs/guides/completion/prompt-design to get more info on the capabilities of the model.

You can either do this homework using the free to use playground/chat interface of openai using the following links:

- https://platform.openai.com/playground
- https://chat.openai.com

But if you want to use the API to make automatic calls to open ai, we will need to follow the steps below:

In [38]:
```
pip install openai
```

```
DEPRECATION: Loading egg at /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/hylaa-0+untagged.602.ga774f8e-py3.11.egg is deprecated. pip 24.3 wi
ll enforce this behaviour change. A possible replacement is to use pip for package installation.. Discussion can be found at https://github.com/pypa/pip/issues/12330
Requirement already satisfied: openai in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (1.3.5)
Requirement already satisfied: anyio<4,>=3.5.0 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from openai) (3.7.1)
Requirement already satisfied: distro<2,>=1.7.0 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from openai) (1.8.0)
Requirement already satisfied: httpx<1,>=0.23.0 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from openai) (0.25.2)
Requirement already satisfied: pydantic<3,>=1.9.0 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from openai) (2.4.2)
Requirement already satisfied: tqdm>4 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from openai) (4.66.1)
Requirement already satisfied: typing-extensions<5,>=4.5 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from openai) (4.8.0)
Requirement already satisfied: idna>=2.8 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from anyio<4,>=3.5.0->openai) (3.4)
Requirement already satisfied: sniffio>=1.1 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from anyio<4,>=3.5.0->openai) (1.3.0)
Requirement already satisfied: certifi in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from httpx<1,>=0.23.0->openai) (2023.7.22)
Requirement already satisfied: httpcore==1.* in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from httpx<1,>=0.23.0->openai) (1.0.2)
Requirement already satisfied: h11<0.15,>=0.13 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from httpcore==1.*->httpx<1,>=0.23.0->opena
i) (0.14.0)
Requirement already satisfied: annotated-types>=0.4.0 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from pydantic<3,>=1.9.0->openai) (0.
5.0)
Requirement already satisfied: pydantic-core==2.10.1 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from pydantic<3,>=1.9.0->openai) (2.1
0.1)

[notice] A new release of pip is available: 23.3 -> 23.3.1
[notice] To update, run: pip3.11 install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import os

## Find the API key by clicking on your profile in the openai page. Add the key to the environment as following:
## Make sure to delete this cell afterwords

os.environ['OPENAI_API_KEY'] = '...'
```

In [3]:
```
from openai import OpenAI
client = OpenAI()

client.api_key = os.getenv('OPENAI_API_KEY')
```

Using text completion

In [4]:
```
response = client.completions.create(
  model="text-davinci-002",
  prompt="Decide whether a Tweet's sentiment is positive, neutral, or negative.\n\nTweet: \"I loved the new Batman movie!\"\nSentiment:",
  temperature=0,
  max_tokens=60,
  top_p=1,
  frequency_penalty=0.5,
  presence_penalty=0
)
```

In [5]:
```
response
```

Out[5]:
```
Completion(id='cmpl-8QOUEoPOrHPVnDRZcKh12O4RcUiJv', choices=[CompletionChoice(finish_reason='stop', index=0, logprobs=None, text=' Positive')], created=1701301346, model='tex
t-davinci-002', object='text_completion', system_fingerprint=None, usage=CompletionUsage(completion_tokens=1, prompt_tokens=31, total_tokens=32), warning='This model version
is deprecated. Migrate before January 4, 2024 to avoid disruption of service. Learn more https://platform.openai.com/docs/deprecations')
```

In [6]:
```
response.choices[0].text
```

Out[6]:
```
' Positive'
```

Using chat completion

In [7]:
```
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "system", "content": "You are a Sentiment Classifier."},
    {"role": "user", "content": "Decide whether a Tweet's sentiment is positive, neutral, or negative.\n\nTweet: \"I loved the new Batman movie!\"\nSentiment:"}
  ]
)
```

In [8]:
```
response
```

Out[8]:
```
ChatCompletion(id='chatcmpl-8QOUKIdzvO8hPjrR86BfDJfogvxla', choices=[Choice(finish_reason='stop', index=0, message=ChatCompletionMessage(content='Positive', role='assistant',
function_call=None, tool_calls=None))], created=1701301352, model='gpt-3.5-turbo-0613', object='chat.completion', system_fingerprint=None, usage=CompletionUsage(completion_to
kens=1, prompt_tokens=46, total_tokens=47))
```

In [9]:
```
response.choices[0].message.content
```

Out[9]:
```
'Positive'
```

If you see ' Positive' as response in the above cell, you have successfully set-up gpt3 in your system.

Now, the task for the assignment is really just do something cool. For example, you could probe how well GPT3 performs on the tasks in the previous HWs. Or, you could do something like question-answering or summarization, that were not covered in the assignments. The choice is yours.

Summarization

In [10]:
```
def summarize(text):
    response = client.chat.completions.create(
      model="gpt-3.5-turbo",
      messages=[
        {"role": "system", "content": "You are a text summarizer"},
        {"role": "user", "content": text}
      ]
    )
    return response
```

In [12]:
```
abstract ="In the past three years, the so-called second wave of Virtual Reality (VR) has brought us a vast amount of new displays and input devices. Not only new hardware has
summary = summarize(abstract)
print(summary.choices[0].message.content)
```

```
This paper discusses the recent advancements in Virtual Reality (VR) technology, focusing on the second wave of VR that has occurred in the past three years. It highlights the
introduction of new displays and input devices, as well as the development of new technologies and concepts for handling existing problems. The development of VR hardware and
software is predominantly led by enthusiasts, rather than the established scientific community. The paper also explores various devices that are gaining importance in the fiel
d of VR, such as haptics devices, controllers, treadmills, and tracking technologies. It emphasizes that these technologies are now precise and robust enough for professional
and scientific use. The paper further addresses common issues with VR technologies, including motion-to-photon latency, barrel distortion, and low-persistence displays, and di
scusses the approaches to solving these problems. It concludes with an analysis of the available solutions and provides a taxonomy categorizing the current developments in VR.
Overall, the paper provides a comprehensive overview of the recent developments in VR technology and considers the future advancements in both hardware and software.
```

In [14]:
```
def summarize(text):
    response = client.chat.completions.create(
      model="gpt-3.5-turbo",
      messages=[
        {"role": "system", "content": "You are a text summarizer"},
        {"role": "user", "content": f"Summarize the following: {text}"}
      ]
    )
    return response
```

In [15]:
```
abstract ="In the past three years, the so-called second wave of Virtual Reality (VR) has brought us a vast amount of new displays and input devices. Not only new hardware has
summary = summarize(abstract)
print(summary.choices[0].message.content)
```

```
The past three years have seen significant advancements in Virtual Reality (VR), with new hardware and technologies being developed. Enthusiasts are leading the way in this do
main, with new concepts and solutions constantly being introduced. In addition to Head-Mounted Displays (HMDs), other devices like haptics devices, controllers, and tracking t
echnologies are gaining importance. These technologies are already precise enough for professional and scientific use. Common issues with these technologies, such as motion-to
-photon latency and barrel distortion, are being addressed, and solutions are expected to hit the market soon. This paper provides a comprehensive overview of the current adva
ncements in VR technology, including upcoming devices and software advancements.
```

POS Tagger

In [16]:
```
def pos(text):
    response = client.chat.completions.create(
      model="gpt-3.5-turbo",
      messages=[
        {"role": "system", "content": "You are a pos tagger"},
        {"role": "user", "content": text}
      ]
    )
    return response
```

In [17]:
```
text = "Shoot all the blue jays you want, if you can hit em, but remember that it’s a sin to kill a mockingbird."
tagged = pos(text)
print(tagged.choices[0].message.content)
```

```
Shoot/VB all/DT the/DT blue/JJ jays/NNS you/PRP want/VBP ,/, if/IN you/PRP can/MD hit/VB em/PRP ,/, but/CC remember/VB that/IN it/PRP ’/VBZ s/VBZ a/DT sin/NN to/TO kill/VB a/D
T mockingbird/NN ./.
```

In [18]:
```
def pos(text):
    response = client.chat.completions.create(
      model="gpt-3.5-turbo",
      messages=[
        {"role": "system", "content": "You are a pos tagger given the tags \nADJ: adjective\nADP: adposition\nADV: adverb\nAUX: auxiliary\nCCONJ: coordinating conjunction\nDE
        {"role": "user", "content": text}
      ]
    )
    return response
```

In [19]:
```
text = "Shoot all the blue jays you want, if you can hit em, but remember that it’s a sin to kill a mockingbird."
tagged = pos(text)
print(tagged.choices[0].message.content)
```

```
Shoot/VERB all/DET the/DET blue/ADJ jays/NOUN you/PRON want/VERB ,/PUNCT if/SCONJ you/PRON can/VERB hit/VERB em/PART ,/PUNCT but/CCONJ remember/VERB that/ADP it/PRON ’s/AUX a/
DET sin/NOUN to/ADP kill/VERB a/DET mockingbird/NOUN ./PUNCT
```

Irony Detection

In [20]:
```
def irony(text):
    response = client.chat.completions.create(
      model="gpt-3.5-turbo",
      messages=[
        {"role": "system", "content": "Is there irony present in the text"},
        {"role": "user", "content": text}
      ]
    )
    return response
```

In [21]:
```
text = "Shoot all the blue jays you want, if you can hit em, but remember that it’s a sin to kill a mockingbird."
entities = irony(text)
print(entities.choices[0].message.content)
```

```
Yes, there is irony present in the text. The irony lies in the fact that while it is acceptable to shoot blue jays, it is considered a sin to kill a mockingbird. The mockingbi
rd is used as a metaphor for innocence and harmlessness, as it is a bird that only sings and does not harm anyone. By stating the sinfulness of killing a mockingbird, the spea
ker is implying that it is wrong to harm something or someone that is innocent and harmless.
```

In [22]:
```
text = "To Kill a Mockingbird is a novel by the American author Harper Lee. It was published in 1960 and was instantly successful. In the United States, it is widely read in
result = irony(text)
print(result.choices[0].message.content)
```

```
There is no irony present in this text.
```

In [23]:
```
def irony(text):
    response = client.chat.completions.create(
      model="gpt-3.5-turbo",
      messages=[
        {"role": "system", "content": "You are an irony detection system"},
        {"role": "user", "content": f"Is there irony present in: {text}"}
      ]
```

```
    )
    return response
```

In [24]:
```
text = "Shoot all the blue jays you want, if you can hit em, but remember that it’s a sin to kill a mockingbird."
entities = irony(text)
print(entities.choices[0].message.content)
```

```
Yes, there is irony present in the statement. The irony lies in the juxtaposition of shooting blue jays, which is allowed and seemingly encouraged, while highlighting the sinf
ulness of killing a mockingbird. Mockingbirds are innocent creatures that only bring joy through their songs, thus the statement is ironic because it suggests a contradiction
between the treatment of different species of birds.
```

In [25]:
```
text = "To Kill a Mockingbird is a novel by the American author Harper Lee. It was published in 1960 and was instantly successful. In the United States, it is widely read in h
result = irony(text)
print(result.choices[0].message.content)
```

```
No, there is no irony present in the provided statement. It simply states information about the novel "To Kill a Mockingbird" and its success.
```

Natural Language Inference

In [42]:
```
def nli(s1, s2):
    response = client.chat.completions.create(
      model="gpt-3.5-turbo",
      messages=[
        {"role": "user", "content": f"Premise: {s1}\n Hypothesis 2: {s2}"}
      ]
    )
    return response

response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "system", "content": "Given two sentences: a premise and a hypothesis, you need to classify the relation between them. We have three classes to describe this rela
  ]
)
```

In [43]:
```
premises=["A man inspects the uniform of a figure in some East Asian country.", "A black race car starts up in front of a crowd of people.", "A soccer game with multiple males
hypothesi=["The man is sleeping", "A man is driving down a lonely road.", "Some men are playing a sport."]

for i in range(len(premises)):
    inference = nli(premises[i], hypothesi[i])
    print(inference.choices[0].message.content)
    print()
```

```
It is important to note that hypothesis 2 cannot be derived from the given premise. The premise only mentions a man inspecting the uniform of a figure in some East Asian count
ry. There is no information provided about the man's state of consciousness or whether he is sleeping. Therefore, hypothesis 2 is not supported by the given premise.

The hypothesis provided does not directly relate to the given premise, which focuses on a black race car starting up in front of a crowd of people. It introduces a different s
cenario involving a man driving down a lonely road.

Your hypothesis is not specific enough to be proven or disproven based on the given premise. The premise states that a soccer game with multiple males playing is taking place,
but it does not provide enough information to determine if the sport being played is solely soccer or if there are other sports involved as well. Additionally, the premise doe
s not explicitly state the gender of all the players, only mentioning that there are multiple males playing. Therefore, it is not possible to validate or invalidate the hypoth
esis solely based on the given premise.
```

In [32]:
```
def nli(s1, s2):
    response = client.chat.completions.create(
      model="gpt-3.5-turbo",
      messages=[
        {"role": "system", "content": "Given two sentences: a premise and a hypothesis, you need to classify the relation between them and provide an explanation. We have thre
        {"role": "user", "content": f"Premise: {s1}\n Hypothesis 2: {s2}"}
      ]
    )
    return response
```

In [34]:
```
premises=["A man inspects the uniform of a figure in some East Asian country.", "A black race car starts up in front of a crowd of people.", "A soccer game with multiple males
hypothesi=["The man is sleeping", "A man is driving down a lonely road.", "Some men are playing a sport."]

for i in range(len(premises)):
    inference = nli(premises[i], hypothesi[i])
    print(inference.choices[0].message.content)
    print()
```

```
Class: Neutral

Explanation: The two sentences do not have a clear relationship. The premise talks about a man inspecting a uniform, while the hypothesis states that the man is sleeping. Ther
e is no inherent contradiction or entailment between the two statements. They can coexist without affecting each other.

Classification: Neutral

Explanation: The premise and the hypothesis do not have any clear relationship. The premise mentions a race car starting up in front of a crowd, while the hypothesis talks abo
ut a man driving down a lonely road. These two sentences describe different scenarios and do not support or contradict each other.

Relation: Entailment

Explanation: The premise states that there is a soccer game with multiple males playing. The hypothesis states that some men are playing a sport. Since soccer is a sport and t
he premise explicitly mentions males playing it, the hypothesis follows from the fact that the premise is true.
```

In [37]:
```
def nli(s1, s2):
    response = client.chat.completions.create(
      model="gpt-3.5-turbo",
      messages=[
        {"role": "system", "content": "Given two sentences: a premise and a hypothesis, you need to classify the relation between them. We have three classes to describe this
        {"role": "user", "content": f"Premise: {s1}\n Hypothesis 2: {s2}"}
      ]
    )
    return response
```

In [38]:
```
premises=["A man inspects the uniform of a figure in some East Asian country.", "A black race car starts up in front of a crowd of people.", "A soccer game with multiple males
hypothesi=["The man is sleeping", "A man is driving down a lonely road.", "Some men are playing a sport."]

for i in range(len(premises)):
    inference = nli(premises[i], hypothesi[i])
    print(inference.choices[0].message.content)
    print()
```

```
Contradiction

Relation: Neutral

Entailment
```

Submission

Please submit a written report of what task you tried probing, how well did GPT3 do for that task and what were your key takeaways in this experiment.

I have conducted an in-depth exploration of the ChatGPT API and experimented with various applications. In an attempt to fulfill assignments, I utilized ChatGPT for diverse tasks, achieving varying degrees of success.

- Summarisation

The model exhibited notable proficiency in summarizing text. Specifically, I endeavored to summarize the abstract of a paper by employing the model, resulting in a satisfactory outcome. My experimentation involved employing two distinct prompts, each yielding commendable results. One prompt solely utilized the system's message, while the other incorporated the task description within the message.

- POS Tagging

  Initially, I refrained from providing specific tags for POS tagging. This approach yielded an unconventional yet comprehensible form of POS tags. Subsequently, when supplied with explicit tags, the model excelled, generating accurate POS tags for all the words in the given context.

- Irony detection

  The model demonstrated proficient performance in detecting irony within sentences, successfully discerning instances of both irony and its absence. Similar to the summarization task, I provided the task in both the system message and the user message, resulting in effective irony detection.

- NLI

  Unfortunately, the model exhibited limitations in its ability to perform Natural Language Inference (NLI) to a satisfactory extent. While experimenting with various prompts and system messages, I identified three approaches that appeared most promising. Due to the presence of multiple tags, additional test cases were essential. However, it is noteworthy that OpenAI imposed a constraint of three prompts per minute. Ultimately, I achieved a moderately acceptable outcome by shortening the system message and omitting the explanation of the tags.

In summary, my exploration of the ChatGPT API revealed its effectiveness in tasks such as text summarization, POS tagging, and irony detection. The model successfully condensed information, accurately tagged parts of speech, and discerned irony in sentences. Challenges emerged in Natural Language Inference (NLI), where the model displayed limitations, though experimentation identified more effective approaches. While the ChatGPT API demonstrated commendable capabilities, ongoing refinement may be necessary, particularly in nuanced applications like NLI, to further enhance its effectiveness.