Introduction

Welcome to this handbook where we embark on a summarization task with the OpenAI API. Our journey begins with setting up all required libraries and diving into the dataset for dialogue summarization. Drawing from our previous learnings, we'll employ prompt engineering to summarize text from the dataset. Lastly, we'll predict and evaluate the LLM's performance to assess its practical utility.

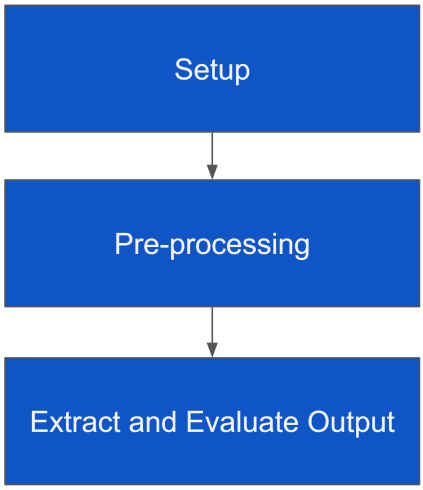
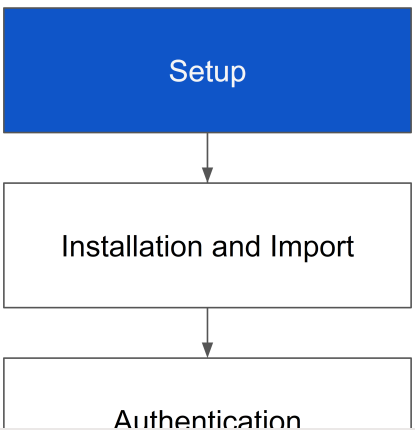


Figure 1

1. Setup

Section Overview: *To begin, we'll first set up the required code for the summarization task. We'll be installing and importing all the necessary libraries and authenticate credentials for the OpenAI API.*



1.1 Installation and Import

Section Overview: *The provided code snippet sets up the Python environment for a summarization tas using the Azure OpenAI API and data processing libraries. It installs necessary packages, imports essential modules for data handling, and model output. This setup prepares the environment for loadin datasets, preprocessing data, training models, and evaluating performance metrics effectively.*

Python----------

```
!pip install -q openai==1.2.0 datasets evaluate rouge_score bert_score
```

```
import pandas as pd
from openai import AzureOpenAI
from datasets import load_dataset
from evaluate import load
from tqdm import tqdm
import json
```

Python----------

- **Install Libraries:** We will install all the necessary libraries here.
  - `!pip install -q openai==1.2.0 datasets evaluate rouge_score bert_score`: This line installs the required packages for the project. It installs the openai, datasets, evaluate, rouge_score, and bert_score packages. rouge_score and bert_score will be metrics we will use later for evaluation.

- **Import Libraries:** The code then imports the libraries that were installed. Most of the libraries imported here are already discussed previously. Let us discuss the libraries we are importing for the first time
  - `import pandas as pd`: We will import the pandas library here. It is useful library to manipulate and analyze data in dataframe format. 'pd' here is an alias we will use for calling pandas library later in code.
  - `from openai import AzureOpenAI`: This line imports the AzureOpenAI class from the openai library, which allows interaction with the Azure OpenAI API.
  - `from datasets import load_dataset`: This line imports the `load_dataset` function

- `from tqdm import tqdm`: This line imports the `tqdm` function from the `tqdm` library.The `tqdm` function is needed to display a progress bar, which can be useful for tracking the progress of long-running tasks.This will help us display progress of responses generated for the dataset.
- `import json`: This line imports the json library, which is used for working with JSON data. Here we will be using json to read the azure credentials file.

1.2 Authentication

Section Overview: *This code snippet is used for OpenAI authentication we have already discussed previously. This code snippet is structured to read authentication credentials and configuration details from a JSON file, initialize an AzureOpenAI client with the provided credentials, and extract the deployment name for a summarizatio task. By loading credentials from an external file, the code ensure secure handling of sensitive information and allows for easy configuration changes without modifying t code directly. The extracted information is then used to set up the AzureOpenAI client for interacting with the Azure OpenAI API, facilitating summarization tasks with the specified model and endpoint.*

Python----------

```
with open('config.json', 'r') as az_creds:
 data = az_creds.read()

creds = json.loads(data)

client = AzureOpenAI(
 azure_endpoint=creds["AZURE_OPENAI_ENDPOINT"],
 api_key=creds["AZURE_OPENAI_KEY"],
 api_version=creds["AZURE_OPENAI_APIVERSION"]
)

deployment_name = creds["CHATGPT_MODEL"]
```

Python----------

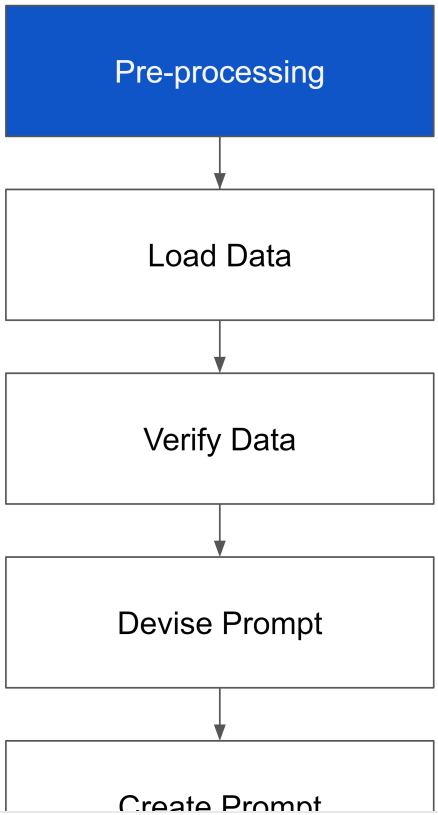
- **Upload and Read Configuration File:** Here we will read the config.json file. This is the same file we created in previous week of the course (Prompt Engineering at Scale) in the API Setup Hands-on

- **Credentials Assignment:**
  - Assign the values from the `creds` dictionary to the corresponding attributes of the `openai` module.These credentials are read by the openai function and act as authentication step.
  - `client = AzureOpenAI(...)`: This is the function that creates an instance of the AzureOpenAI class. This class is used to authenticate to the personalized Open AI model server and make requests to the API.

- **Deployment Name:**
  - `deployment_name = creds["CHATGPT_MODEL"]`: Assign the deployment name of the cha completion endpoint to the **deployment_name** variable.
  - This name is used to specify which chat model should be accessed when making requests to the Azure OpenAI API.

2. Pre-Processing

Section Overview: *With all the necessary requirements in place, we can now proceed to load and preprocess the data. We'll ensure it's formatted appropriately for input to the OpenAI API. Additionally, we'll set aside separate gold examples to be used for prediction later. Finally, we will create a prompt which will act as an instruction to the OpenAI API for the output generation.*



2.1 Load Data

Section Overview: *The provided code loads the dataset, which contains dialogue conversation and summary of the conversation. We will then extract the examples and gold examples from the data. Examples will be provided in input prompt and gold examples will be used for evaluation.*

Python----------

```
# Load the dataset
dataset = load_dataset("pgurazada1/dialogsum")
dialogue_summary_examples_df = dataset['examples'].to_pandas()
dialogue_summary_gold_examples_df = dataset['gold_examples'].to_pandas()
```

```
examples_df = dialogue_summary_examples_df.sample(4)
examples_df
```

	dialogue	summary
29	#Person1#: Are you free on the thirteenth in t...	#Person1# and #Person2# are arranging for thei...
2	#Person1#: What is the best way to find a job ...	#Person2# tells #Person1# to check the binder ...
0	#Person1#: Have you gone to school today?\n#Pe...	#Person1# couldn't go to school for the illnes...
12	#Person1#: I don't understand why you always l...	#Person2# tells #Person1# #Person2# always doe...

Python----------

- **Load Dataset:**
  - `dataset = load_dataset("pgurazada1/dialogsum")`: This line loads the dataset named "dialogsum" from the user "pgurazada1" using the load_dataset function. It retrieves the dataset containing dialogue summarization examples.
  - `dialogue_summary_examples_df = dataset['examples'].to_pandas()`: This line extracts the 'examples' portion of the dataset loaded in the previous step and converts it into a pandas DataFrame named dialogue_summary_examples_df. This DataFrame will contain the dialogue summarization examples.
  - `dialogue_summary_gold_examples_df = dataset['gold_examples'].to_pandas()`: This line extracts the 'gold_examples' portion

analysis or testing.

- `examples_df`: Finally, this line simply displays the examples_df DataFrame, showing the 4 randomly selected dialogue summarization examples for further processing or analysis.

2.2 Verify Data

Section Overview: *The provided section allows you to visually inspect the dialogue and summary examples to ensure that the data is correctly formatted and matches your expectations. By printing the examples, you can quickly identify any issues or anomalies in the data, such as missing values, incorrec formatting, or unexpected content.*

Python----------

```
# assign sentiment from number to label
for index, row in examples_df.iterrows():
 print('Example Dialogue:')
 print(row[0])
 print('Example Summary:')
 print(row[1])
 break
```

```
Example Dialogue:
#Person1#: Are you free on the thirteenth in the afternoon?
#Person2#: No I'm afraid not. I'm meeting Ruth then. How about the fourteenth in the morning?
#Person1#: I'm sorry. I'm attending a meeting at the Hilton then.
#Person2#: What about the next day?
#Person1#: No. I'm busy then too. I'm meeting Dorothy Heath at North Bridge Road. Are you free on Thursday afternoon?
#Person2#: Yes, I think I am. Let's meet for lunch at mouth restaurant.
#Person1#: Good idea! Is two o'clock okay?
#Person2#: That's fine. See you there!
Example Summary:
#Person1# and #Person2# are arranging for their next meeting. They decide to meet on Thursday afternoon.
```

Python----------

- **Verify Data:**
  - `for index, row in examples_df.iterrows()`: This line starts a for loop that iterates over each row in the examples_df DataFrame. The iterrows() method returns both the index label and a Series containing the data in each row.
  - `print('Example Dialogue:')`: This line prints the string "Example Dialogue:" to separate the dialogue from the summary in the output.

- `print(row)`: This line prints the summary example from the current row of the DataFrame. The row syntax accesses the second column (index 1) of the current row, which contains the summary text.
- `break`: This line breaks out of the for loop after printing the first example. It is included here for demonstration purposes, but in a real scenario, you would likely want to print all the examples to thoroughly verify the data.

2.3 Devise Prompt

Section Overview: *Now, we will create a few-shot prompt which will be used as an input to the OpenAI API function for the summarization task. Here we are using few shot prompt to make the OpenAI familia with the type of input and output expected.*

By setting up this prompt, you are establishing the context and expectations for the summarization task When the AI system receives the user input, it will use this prompt as a guide to generate a concise and specific summary of the dialogue.This approach allows for more targeted and controlled responses from the AI system.

Python----------

```
system_message = """
Summarize the dialogue mentioned in the user input. Be specific and concise in your summary.
"""


few_shot_prompt = [{'role':'system', 'content': system_message}]
```

Python----------

- **Devise Prompt**:
  - `system_message = """...."""`: This line starts a multi-line string assignment for the system_message variable. The triple quotes allow for a more readable and flexible way to define the string.
  - `few_shot_prompt = [{'role':'system', 'content': system_message}]`: This line creates a list containing a single dictionary. The dictionary has two keys: 'role' and 'content'. The 'role' key is set to 'system', indicating that this is a system message. The 'content' key is assigned the value of the system_message variable, which contains the instructions for the summarization task.

by adding user input examples and corresponding assistant output examples to the final prompt.

Python----------

```
for index, row in examples_df.iterrows():
 user_input_example = row[0]
 assistant_output_example = row[1]

 few_shot_prompt.append(
 {
 'role': 'user',
 'content': user_input_example
 }
 )

 few_shot_prompt.append(
 {
 'role': 'assistant',
 'content': assistant_output_example
 }
 )
```

```
few_shot_prompt
```

Python----------

- **Create Prompt**:
  - `for index, row in examples_df.iterrows()`: This line starts a for loop that iterates over each row in the examples_df DataFrame.
  - `user_input_example = row[0]`: This method randomly selects 50 rows from the DataFrame for creating gold examples. The `random_state=42` ensures reproducibility of th random sampling.
  - `assistant_output_example = row[1]`: This line extracts the assistant output example from the current row of the DataFrame and assigns it to the assistant_output_example variable. The row syntax accesses the second column (index 1) of the current row, which contains the assistant output.

example.

- `{'role': 'assistant', 'content': assistant_output_example}`: This dictionary represents the assistant output example. It has a 'role' key set to 'assistant' and a 'content' ke containing the assistant output example.

By iterating through the examples in the DataFrame, extracting the user input and assistant output for each example, and adding them to the few_shot_prompt list with their respective roles, this code effectively constructs a few-shot prompt that includes both user input and expected assistant output examples.

3. Extract and Evaluate Output

Section Overview: *Let's dive into the process of generating and evaluating results for the summarizatic task. For evaluation, we'll employ two metrics: ROUGE and BERTScore. Further details on these evaluation metrics can be found on the following page.*

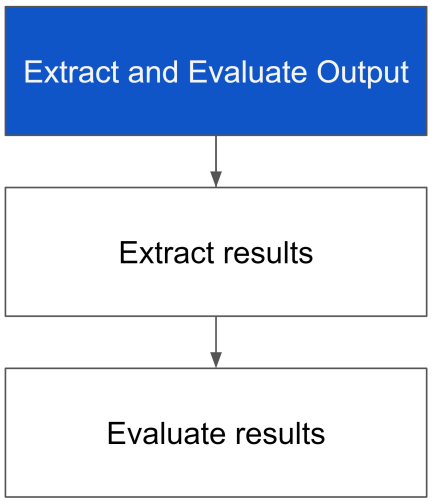


Figure 4

3.1 Extract Output

Section Overview: *Let us do the output generation here for the the gold examples. We will send reques to API here to generate summary for us.*

Python----------

```
predictions, ground_truths = [], []
```

```
user_input = [{'role':'user', 'content': gold_dialogue}]

try:
response = client.chat.completions.create(
model=deployment_name,
messages=few_shot_prompt + user_input
)

predictions.append(response.choices[0].message.content)
ground_truths.append(gold_summary)
except Exception as e:
print(e) # Log error and continue
continue
```

Python----------

- **User Message**:
  - `predictions, ground_truths = [], []` : This line initializes two empty lists, predictions and ground_truths, which will store the predicted summaries and the actual (ground truth) summaries, respectively.
  - `for index, row in tqdm(dialogue_summary_gold_examples_df.iterrows())` : This line starts a loop that iterates over each row in the dialogue_summary_gold_examples_df DataFrame, which contains the gold standard dialogue and summary examples.
  - `gold_dialogue = row and gold_summary = row` : These lines extract the gold dialogue and gold summary from the current row of the DataFrame.
  - `user_input = [{'role':'user', 'content': gold_dialogue}]` : This line creates a user input dictionary containing the gold dialogue. This user input will be used as part of the input to the AI model.
  - `try` : The try block in Python is used to test a block of code for errors. It allows you to handl exceptions that may occur during the execution of the code. Here we are using this to catch the exception (if any) given by the chat completion function.
  - `response = client.chat.completions.create(...)` : This line sends a request to the Azure OpenAI API to generate completions (summaries) based on the provided input.
  - `model=deployment_name, messages=few_shot_prompt + user_input)` : This part of the request specifies the model to use (deployment_name) and combines the few-shot prompt and user input to provide context for the AI model.
  - `predictions.append(response.choices.message.content)` : This line appends the

`except Exception as e` : If an exception occurs during the API call, this block of code will catch the exception.

- `print(e)` : This line prints the error message for logging purposes.
- `continue` : This keyword continues to the next iteration of the loop if an exception occurs, ensuring that the process continues despite errors.

3.2 Evaluate

Section Overview: *The code here provided is performing evaluation using the BERTScore and ROUGE metrics.By calculating these evaluation metrics, you can assess the performance of your summarization model and compare it to other models or baselines. The results can help you identify areas for improvement and guide further development of your summarization system.*

NOTE: *The BERTScore and ROUGE metrics provide a quantitative way to evaluate the quality of the generated summaries by comparing them to the ground truth summaries. The BERTScore measures semantic similarity, while ROUGE measures the overlap of n-grams between the predicted and ground truth summaries. You can read more about the evaluation metrics in the next section of this module.*

Python----------

```
bertscore = load("bertscore")
rouge_scorer = load("rouge")
```

```
results = bertscore.compute(
 predictions=predictions,
 references=ground_truths,
 lang="en"
)

sum(results['f1'])/len(results['f1'])
```

```
0.8962427414953709
```

```
rouge_scorer.compute(
 predictions=predictions,
 references=ground_truths,
 )
```

Python----------

- **Load Evaluation Metrics**:
  - `bertscore = load("bertscore")` : This line loads the BERTScore evaluation metric fror the evaluate library. BERTScore is a metric that compares the similarity between the predicte summaries and the ground truth summaries using BERT embeddings.
  - `rouge_scorer = load("rouge")` : This line loads the ROUGE evaluation metric from the evaluate library. ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a set of metric that compare the overlap between the predicted summaries and the ground truth summaries

- **Evaluate:**
  - `results = bertscore.compute(predictions=predictions, references=ground_truths, lang="en")` : This line computes the BERTScore for each predicted summary and ground truth summary pair. It takes the following arguments:
    - `predictions` : The list of predicted summaries
    - `references` : The list of ground truth summaries
    - `lang` : The language of the summaries, in this case, "en" for English
      The `compute` function returns a dictionary containing the BERTScore results.
  - `sum(results['f1'])/len(results['f1'])` : This line calculates the average BERTScore F1 score. It sums up all the F1 scores in the results['f1'] list and divides it by the number of scores to get the mean.
  - `rouge_scorer.compute(predictions=predictions, references=ground_truths)` : This line computes the ROUGE scores for each predicted summary and ground truth summary pair. It takes the following arguments:
    - `predictions` : The list of predicted summaries
    - `references` : The list of ground truth summaries
      The `compute` function returns a dictionary containing the ROUGE scores.