Task 1a: Perform Text Generation

In this notebook, you learn how to use Large Language Model (LLM) to generate an email response to a customer who provided negative feedback on the quality of customer service they received from the support engineer. In this notebook, you generate an email with a thank you note based on the customer's previous email. You use the Amazon Titan model using the Amazon Bedrock API with Boto3 client.

The prompt used in this task is called a zero-shot prompt. In a zero-shot prompt, you describe the task or desired output to the language model in plain language. The model then uses its pre-trained knowledge and capabilities to generate a response or complete the task based solely on the provided prompt.

Scenario

You are Bob a Customer Service Manager at AnyCompany and some of your customers are not happy with the customer service and are providing negative feedbacks on the service provided by customer support engineers. Now, you would like to respond to those customers apologizing for the poor service and to regain trust. You need the help of an LLM to generate a bulk of emails for you which are human friendly and personalized to the customer's sentiment from previous email correspondence.

Task 1a.1: Environment setup

In this task, you set up your environment.

In [ ]:
```
#Create a service client by name using the default session.
import json
import os
import sys

import boto3
import botocore

module_path = ".."
sys.path.append(os.path.abspath(module_path))

bedrock_client = boto3.client('bedrock-runtime',region_name=os.environ.get("AWS_
```

Task 1a.2: Generate text

In this task, you prepare an input for the Amazon Bedrock service to generate an email.

In [ ]:
```
# create the prompt
prompt_data = """
Command: Write an email from Bob, Customer Service Manager, AnyCompany to the cu
```

```
who provided negative feedback on the service provided by our customer support engineer"""
```

In [ ]:
```
body = json.dumps({
    "inputText": prompt_data,
    "textGenerationConfig":{
        "maxTokenCount":8192,
        "stopSequences":[],
        "temperature":0,
        "topP":0.9
        }
    })
```

Next, you use the Amazon Titan model.

Note: Amazon Titan supports a context window of ~4k tokens and accepts the following parameters:

- `inputText` : Prompt to the LLM
- `textGenerationConfig` : These are the parameters that model will take into account while generating the output.

The Amazon Bedrock API provides you with an API `invoke_model` which accepts the following:

- `modelId` : This is the model ARN for the various foundation models available under Amazon Bedrock
- `accept` : The type of input request
- `contentType` : The content type of the output
- `body` : A json string consisting of the prompt and the configurations

Refer to documentation for Available text generation model Ids.

Task 1a.3: Invoke the Amazon Titan Large language model

In this task, you explore how the model generates an output based on the prompt created earlier.

Complete Output Generation

This email is generated using the Amazon Titan model by understanding the input request and utilizing its inherent understanding of different modalities. The request to the API is synchronous and waits for the entire output to be generated by the model.

In [ ]:
```
#invoke model
modelId = 'amazon.titan-text-express-v1' # change this to use a different versio
accept = 'application/json'
contentType = 'application/json'
outputText = "\n"
try:
```

```
    response = bedrock_client.invoke_model(body=body, modelId=modelId, accept=ac
    response_body = json.loads(response.get('body').read())

    outputText = response_body.get('results')[0].get('outputText')

except botocore.exceptions.ClientError as error:

    if error.response['Error']['Code'] == 'AccessDeniedException':
           print(f"\x1b[41m{error.response['Error']['Message']}\
                \nTo troubeshoot this issue please refer to the following resour
                 \nhttps://docs.aws.amazon.com/IAM/latest/UserGuide/troubleshoot
                 \nhttps://docs.aws.amazon.com/bedrock/latest/userguide/security

    else:
        raise error
```

In [ ]:
```
# The relevant portion of the response begins after the first newline character
# Below we print the response beginning after the first occurence of '\n'.

email = outputText[outputText.index('\n')+1:]
print(email)
```

Streaming Output Generation

Bedrock also supports that the output can be streamed as it is generated by the model in form of chunks. This email is generated by invoking the model with streaming option. `invoke_model_with_response_stream` returns a `ResponseStream` which you can read from.

In [ ]:
```
# invoke model with response stream
output = []
try:

    response = bedrock_client.invoke_model_with_response_stream(body=body, model
    stream = response.get('body')

    i = 1
    if stream:
        for event in stream:
            chunk = event.get('chunk')
            if chunk:
                chunk_obj = json.loads(chunk.get('bytes').decode())
                text = chunk_obj['outputText']
                output.append(text)
                print(f'\t\t\x1b[31m**Chunk {i}**\x1b[0m\n{text}\n')
                i+=1

except botocore.exceptions.ClientError as error:

    if error.response['Error']['Code'] == 'AccessDeniedException':
           print(f"\x1b[41m{error.response['Error']['Message']}\
                \nTo troubeshoot this issue please refer to the following resour
                 \nhttps://docs.aws.amazon.com/IAM/latest/UserGuide/troubleshoot
                 \nhttps://docs.aws.amazon.com/bedrock/latest/userguide/security
```

```
    else:
        raise error
```

The stream with response approach helps to quickly obtain the output of the model and allows the service to complete it as you read. This assists in use cases where you request the model to generate longer pieces of text. You can later combine all the chunks generated to form the complete output and use it for your use case.

In [ ]:
```
#combine output chunks
print('\t\t\x1b[31m**COMPLETE OUTPUT**\x1b[0m\n')
complete_output = ''.join(output)
print(complete_output)
```

You have now experimented with using the boto3 SDK, which provides basic exposure to the Amazon Bedrock API. Using this API, you have seen the use case of generating an email to respond to a customer's negative feedback.

Try it yourself

- Change the prompts to your specific usecase and evaluate the output of different models.
- Play with the token length to understand the latency and responsiveness of the service.
- Apply different prompt engineering principles to get better outputs.

Cleanup

You have completed this notebook. To move to the next part of the lab, do the following:

- Close this notebook file and continue with **Task1b.ipynb.**