Customer Complaint Classification

Approach:

For the initial **transcription.txt** file, I have self-recorded an audio file that looks like a customer complaint. The original transcript of the audio as well as the audio file can be found in the directory named audio with the names **customer_complaint.txt** and **customer_complain.wav** respectively. The recorder audio file is then used by the **whisper.py** script to convert the audio into transcribed text.

1. whisper.py:

The main objective of this code is to take an audio file and generate the text from the audio. For this project, I have used the whisper model from OpenAI. The **transcribe_audio()** function present in the **whisper.py** takes azure secrets as well as an audio file as the input and it saves the transcribed audio into a text file at **./output/transcription.txt**. The function returns the transcribed text as a string.

```python
# Function to transcribe customer audio complaints using the Whisper model
def transcribe_audio(azure_secrets: dict, audio_file:str):
    """
    Transcribes an audio file into text using OpenAI's Whisper model.

    Returns:
    str: The transcribed text of the audio file.
    """
    # create openai client
    client = create_openai_client(
        azure_secrets['WHISPER_API_VERSION'],
        azure_secrets['AZURE_API_KEY'],
        azure_secrets['AZURE_ENDPOINT']
    )

    try:
        # Load the audio file.
        with open(audio_file, 'rb') as audio_file:

            # Call the Whisper model to transcribe the audio file.
            transcription = client.audio.transcriptions.create(
                file=audio_file,
                model=azure_secrets['WHISPER_DEPLOYMENT'],
            )
        # save the transcribed audio
        with open('./output/transcription.txt', 'w') as text_file:
            text_file.write(transcription.text)

        print(f"file :{audio_file} transciption completed....")
        print(f"transcription saved to: ./output/transcription.txt")

        # Extract the transcription and return it.
        return transcription.text

    except Exception as e:
        print(f"An error has occured: {e}")
        return None
```

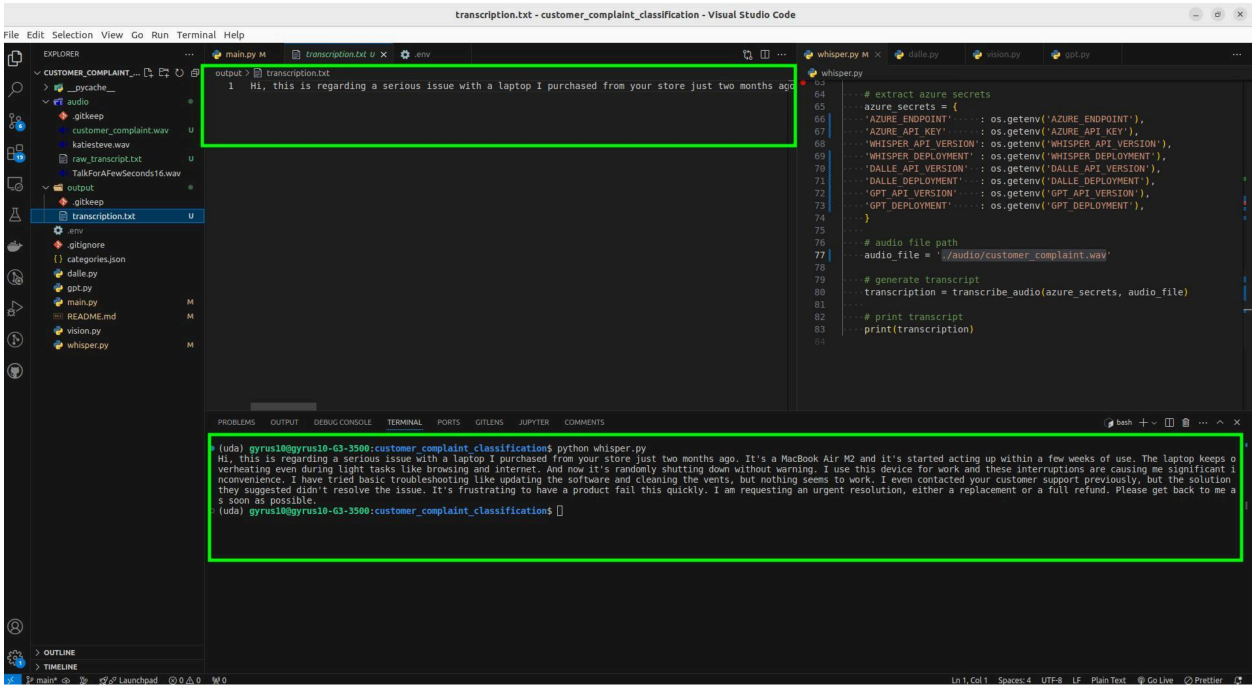
The contents of the audio that was given to the model are:

Hi, this is regarding a serious issue with a laptop I purchased from your store just two months ago. It’s a Macbook Air M2, and it started acting up within a few weeks of use. The laptop keeps overheating even during light tasks like browsing the internet, and now it’s randomly shutting down without warning. I use this device for work, and these interruptions are causing me significant inconvenience. I’ve tried basic troubleshooting like updating the software and cleaning the vents, but nothing seems to work. I even contacted your customer support previously, but the solution they suggested didn’t resolve the issue. It’s frustrating to have a product fail this quickly. I’m requesting an urgent resolution—either a replacement or a full refund. Please get back to me as soon as possible.

Below is the output from the whisper model:

Hi, this is regarding a serious issue with a laptop I purchased from your store just two months ago. It's a MacBook Air M2 and it's started acting up within a few weeks of use. The laptop keeps overheating even during light tasks like browsing and internet. And now it's randomly shutting down without warning. I use this device for work and these interruptions are causing me significant inconvenience. I have tried basic troubleshooting like updating the software and cleaning the vents, but nothing seems to work. I even contacted your customer support previously, but the solution they suggested didn't resolve the issue. It's frustrating to have a product fail this quickly. I am requesting an urgent resolution, either a replacement or a full refund. Please get back to me as soon as possible.

By comparing the original transcript and the transcript generated by the whisper model, we can conclude that the generated transcript is pretty accurate.



```
Hi, this is regarding a serious issue with a laptop I purchased from your store just two months ago
```

```python
    # extract azure secrets
    azure_secrets = {
    'AZURE_ENDPOINT'      : os.getenv('AZURE_ENDPOINT'),
    'AZURE_API_KEY'       : os.getenv('AZURE_API_KEY'),
    'WHISPER_API_VERSION': os.getenv('WHISPER_API_VERSION'),
    'WHISPER_DEPLOYMENT' : os.getenv('WHISPER_DEPLOYMENT'),
    'DALLE_API_VERSION'  : os.getenv('DALLE_API_VERSION'),
    'DALLE_DEPLOYMENT'   : os.getenv('DALLE_DEPLOYMENT'),
    'GPT_API_VERSION'    : os.getenv('GPT_API_VERSION'),
    'GPT_DEPLOYMENT'     : os.getenv('GPT_DEPLOYMENT'),
    }

    # audio file path
    audio_file = './audio/customer_complaint.wav'

    # generate transcript
    transcription = transcribe_audio(azure_secrets, audio_file)

    # print transcript
    print(transcription)
```

```
(uda) gyrus10@gyrus10-G3-3500:customer_complaint_classification$ python whisper.py
Hi, this is regarding a serious issue with a laptop I purchased from your store just two months ago. It's a MacBook Air M2 and it's started acting up within a few weeks of use. The laptop keeps o
verheating even during light tasks like browsing and internet. And now it's randomly shutting down without warning. I use this device for work and these interruptions are causing me significant i
nconvenience. I have tried basic troubleshooting like updating the software and cleaning the vents, but nothing seems to work. I even contacted your customer support previously, but the solution
they suggested didn't resolve the issue. It's frustrating to have a product fail this quickly. I am requesting an urgent resolution, either a replacement or a full refund. Please get back to me a
s soon as possible.
(uda) gyrus10@gyrus10-G3-3500:customer_complaint_classification$ 
```

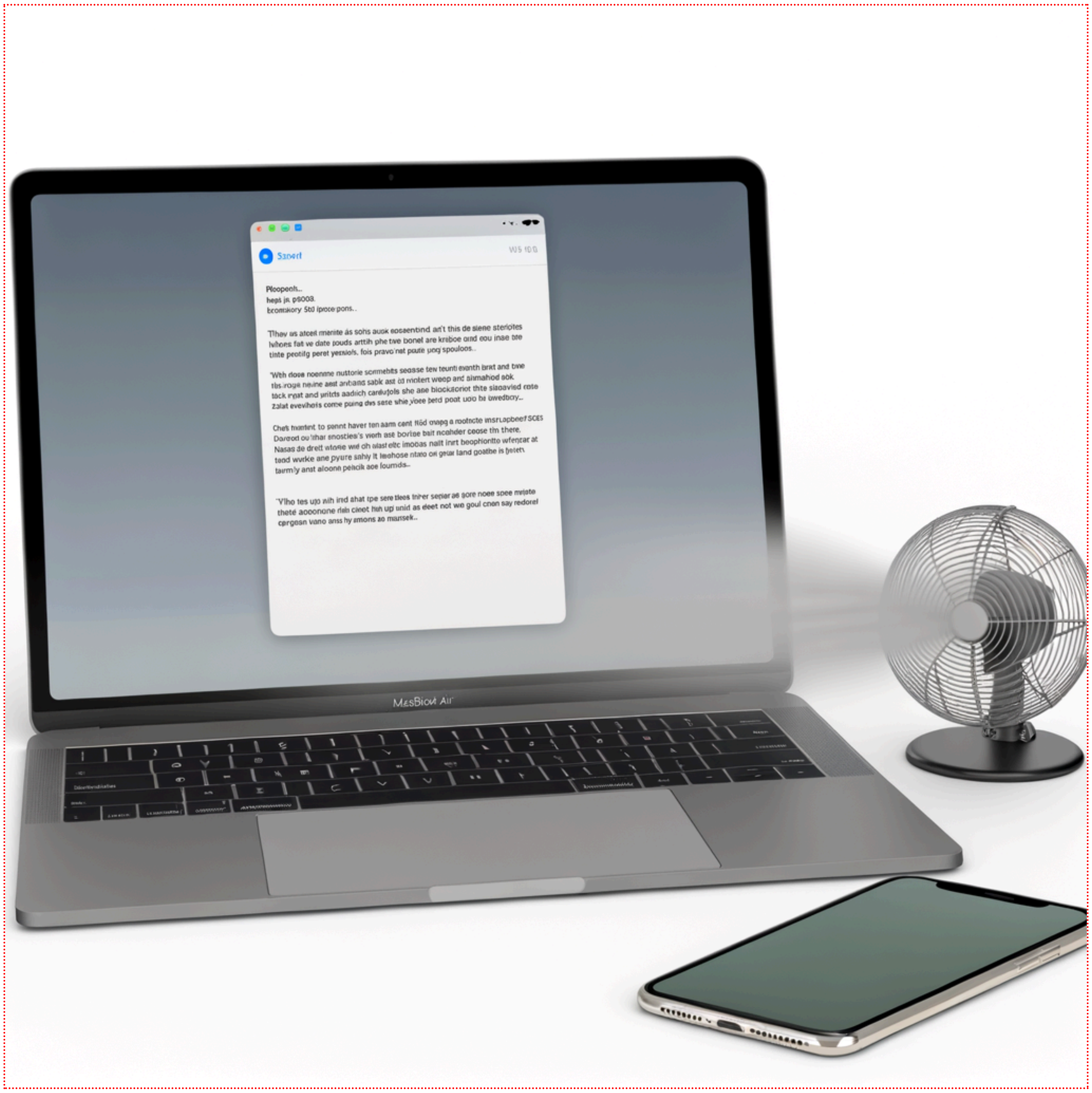
After this step, the generated transcript is used in the dalle.py.

2. dalle.py:

In the dalle.py I have written a function **generate_image()** which takes azure secrets, a prompt containing instructions to generate an image based on the transcribed text, the generated image size, image quality, and tone of the image. This function generates the image and saves it at **./output/generated_image.png** and returns the image path as well as the image URL. I have used the dall-e 3 model for image generation.

```python
# Function to generate an image representing the customer complaint
def generate_image(azure_secrets: dict, prompt: str, size: str, quality: str, style: str):
    """
    Generates an image based on a prompt using OpenAI's DALL-E model.

    Returns:
    str: The path to the generated image.
    """

    # create openai client
    client = create_openai_client(
        azure_secrets['DALLE_API_VERSION'],
        azure_secrets['AZURE_API_KEY'],
        azure_secrets['AZURE_ENDPOINT']
    )

    # Create a prompt to represent the customer complaint.


    # Call the DALL-E model to generate an image based on the prompt.
    result = client.images.generate(
        model=azure_secrets['DALLE_DEPLOYMENT'],
        prompt=prompt,
        size=size,
        quality=quality,
        style=style
    )
    print(f"Image generated....")
    json_response = json.loads(result.model_dump_json())
    image_url = json_response['data'][0]['url']

    # Download the generated image and save it locally.
    image = Image.open(requests.get(image_url, stream=True).raw)
    image_path = './output/generated_image.png'
    image.save(image_path)
    print(f'image saved at: {image_path}')
    print(f'image URL: {image_url}')

    # return image path
    return image_path, image_url
```

The generated image looks like this:



Now this generated image is given to the GPT 4o model for generating the description.

3. vision.py

This Python file uses a function with the name **describe_image()** which takes the image generated from the dalle 3 and a text prompt to generate the image description. The generated description is saved in the **./output/image_description.txt** file. Along with the image description, the prompt for the GPT model is defined such that it identifies the key elements of the image and returns the label and bounding box in a JSON format which will further help us in annotating the key parts of the image. The annotation information is saved in a separate JSON file located at **./output/image_description_annotation.json**. This will help us automate the annotation part by simply loading the annotation from the saved JSON file and loading it for use. The prompt used for generating the description looks like below:

```
description_prompt = """
   Identify key visual elements related to the customer complaint in the
image.
   Return a brief description along with the bounding box details for the
key elements
   in the image in a JSON format with the following structure:

   {
       "description": "<image description>",
       "annotation": [
           {"bbox": (<x_min>, <y_min>, <x_max>, <y_max>), 'label':
<object_in_region>},
           {"bbox": (<x_min>, <y_min>, <x_max>, <y_max>), 'label':
<object_in_region>}
       ]
   }
   """
```

Whereas, the code used to generate the image description looks like this:

```
# Function to describe the generated image and annotate issues
def describe_image(azure_secrets, image_path, prompt):
   """
   Describes an image and identifies key visual elements related to the
customer complaint.

   Returns:
   str: A description of the image, including the annotated details.
   """
   # Load the generated image.
```

```
    data_url = local_image_to_data_url(image_path)

    # crete openai client
    client = create_openai_client(
        azure_secrets['GPT_API_VERSION'],
        azure_secrets['AZURE_API_KEY'],
        azure_secrets['AZURE_ENDPOINT']
    )

    # Call the model to describe the image and identify key elements.
    response = client.chat.completions.create(
        model=azure_secrets['GPT_DEPLOYMENT'],
        response_format={ "type": "json_object" },
        messages=[
            {"role": "system", "content": "You are a helpful assistant designed
to output JSON."},
            {
                "role": "user",
                "content": [
                    {"type": "text", "text": prompt},
                    {"type": "image_url", "image_url": {"url": data_url}}
                ]
            }
        ],
        max_tokens=1024
    )
    image_description = response.choices[0].message.content
    print(image_description)

    desc_json = ast.literal_eval(image_description)

    with open('./output/image_description.txt', 'w') as desc_file:
        desc_file.write(str(desc_json))
    print(f"image description saved to :./output/image_description.txt")

    with open('./output/image_description_annotation.json', 'w') as annot_file:
        json.dump(desc_json, annot_file, sort_keys=True, indent=4)
    print(f"image annotation info saved to:
./output/image_description_annotation.json")

    # Extract the description and return it.
    return desc_json
```

This function also returns the description of the image in JSON format. The sample output of the image description is provided below:

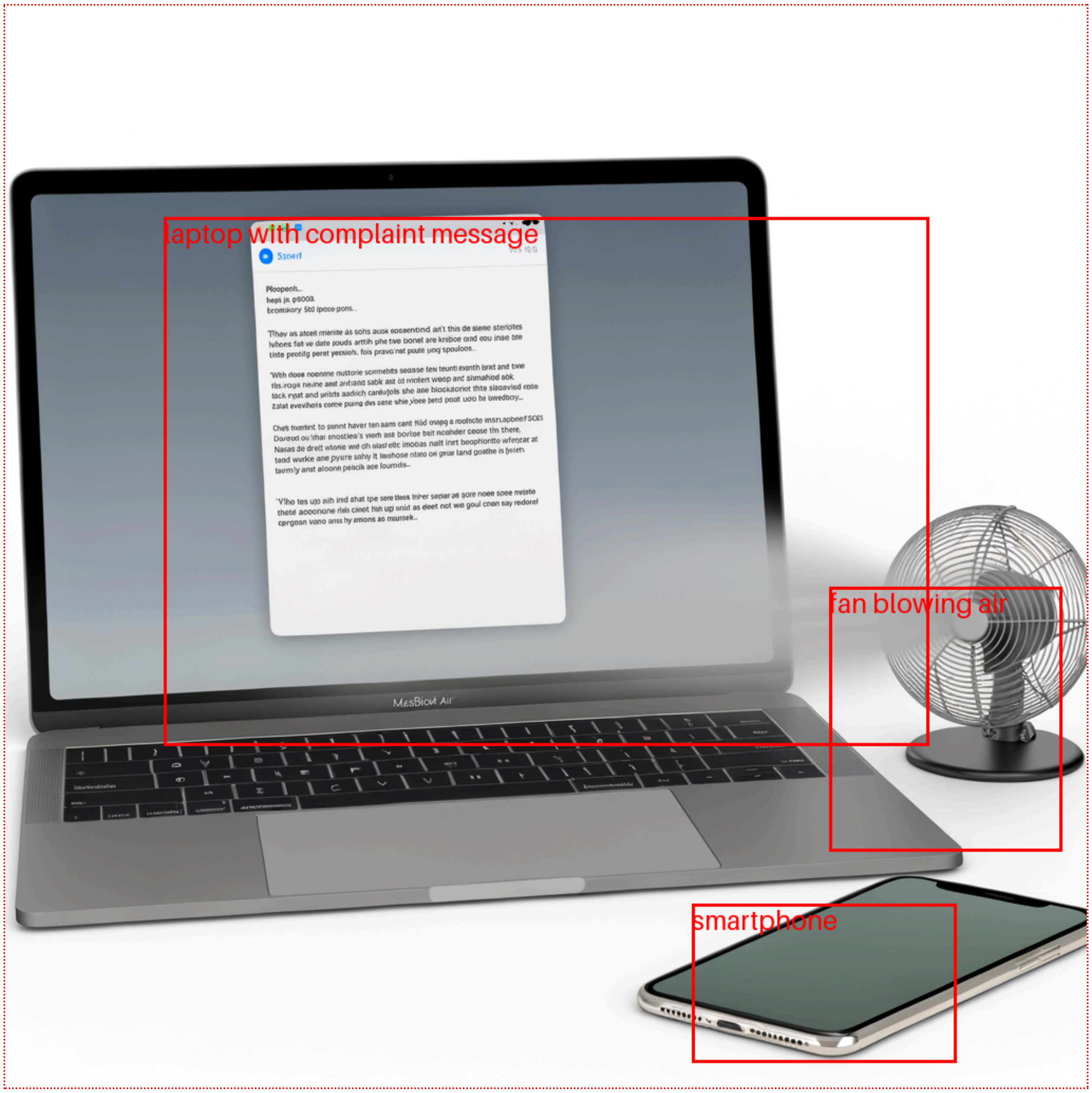
```
{
    "description": "The image shows a laptop with a complaint message on
the screen, a smartphone, and a fan blowing air towards the laptop,
possibly related to electronic overheating issues.",
    "annotation": [
        {
            "bbox": [150, 200, 874, 700],
            "label": "laptop with complaint message"
        },
        {
            "bbox": [780, 550, 1000, 800],
            "label": "fan blowing air"
        },
        {
            "bbox": [650, 850, 900, 1000],
            "label": "smartphone"
        }
    ]
}
```

4. annotate_image

Once the image description is generated, the annotation part from the image description is used to perform the image annotation using the **annotate_image()** function. The function used to perform the annotation can be seen below:

```python
def annotate_image(image_path: str, annotations: list) -> str:
    """
    Draws bounding boxes or annotations on the image and saves it locally.
    Returns the path to the annotated image.
    """
    # response = requests.get(image_url)
    image = Image.open(image_path)

    draw = ImageDraw.Draw(image)
    for annotation in annotations:
        box = annotation['bbox']
        label = annotation['label']
        draw.rectangle(box, outline="red", width=3)
        draw.text((box[0], box[1]), label, fill="red", font_size=25)

    annotated_path = "./output/annotated_image.png"
    image.save(annotated_path)
    return annotated_path
```

The annotated image is saved at **./output/annotated_image.png**. The final annotated image looks something like this.



5. gpt.py()

In gpt.py, I have defined a function **classify_with_gpt()** which takes the image description, and all the categories and subcategories listed in the categories.json file and it classifies the complaint by using the image description into a category and a subcategory. The classification output is saved at **./output/classification.txt** The function looks like this:

```python
# Function to classify the customer complaint based on the image description
def classify_with_gpt(azure_secrets, complaint, categories_file):
   """
   Classifies the customer complaint into a category/subcategory based on the
image description.

   Returns:
   str: The category and subcategory of the complaint.
   """
   # load categories from the json file
   with open(categories_file, 'r') as f:
       categories = json.load(f)

   # Create a prompt that includes the image description and other relevant
details.
   prompt = f"Classify this complaint Description: {complaint}\nProvide
categories and subcategories from below json file.\n {categories}."

   # crete openai client
   client = create_openai_client(
       azure_secrets['GPT_API_VERSION'],
       azure_secrets['AZURE_API_KEY'],
       azure_secrets['AZURE_ENDPOINT']
   )

   # Call the GPT model to classify the complaint based on the prompt.
   response = client.chat.completions.create(
       model=azure_secrets['GPT_DEPLOYMENT'],
       messages=[
           {"role": "system", "content": "You are a classification expert."},
           {
               "role": "user",
               "content": [
                   {"type": "text", "text": prompt},
                   # {"type": "image_url", "image_url": {"url": data_url}}
               ]
           }
       ],
       max_tokens=1024
   )
   # Extract the description and return it.
   classification = response.choices[0].message.content

   # save the classification result
   with open('./output/classification.txt', 'w') as text_file:
        text_file.write(classification)

    print(f"Classification completed....")
    print(f"classification result saved to: ./output/classification.txt")

    return classification
```

Challenges:

The overall project was pretty straight forward and I didn't face any challenges throughout the project. I have tested all the individual Python scripts whisper.py, dalle.py, vision.py, and gpt.py separately by writing the test code and it has worked without throwing any errors.

Finally, I run the **main.py** file which runs the whole application end to end, by taking the audio file as an input and finally providing the classification for the complaint. The output log after running the main file is given below.

```
Transcribing Audio...

file :<_io.BufferedReader name='./audio/customer_complaint.wav'>
transciption completed....
transcription saved to: ./output/transcription.txt
Hi, this is regarding a serious issue with a laptop I purchased from your
store just two months ago. It's a MacBook Air M2 and it's started acting up
within a few weeks of use. The laptop keeps overheating even during light
tasks like browsing and internet. And now it's randomly shutting down
without warning. I use this device for work and these interruptions are
causing me significant inconvenience. I have tried basic troubleshooting
like updating the software and cleaning the vents, but nothing seems to
work. I even contacted your customer support previously, but the solution
they suggested didn't resolve the issue. It's frustrating to have a product
fail this quickly. I am requesting an urgent resolution, either a
replacement or a full refund. Please get back to me as soon as possible.






Generating Image...
```

```
Image generated....
image saved at: ./output/generated_image.png
image URL:
https://dalleprodsec.blob.core.windows.net/private/images/12ead376-8363-487
8-a265-88398224a148/generated_00.png?se=2024-12-14T16%3A18%3A37Z&sig=s0I7TQ
0kXBQvtRNxlgZOs5s95KxhdiRtUJzuUPg244I%3D&ske=2024-12-20T06%3A31%3A15Z&skoid
=e52d5ed7-0657-4f62-bc12-7e5dbb260a96&sks=b&skt=2024-12-13T06%3A31%3A15Z&sk
tid=33e01921-4d64-4f8c-a055-5bdaffd5e33d&skv=2020-10-02&sp=r&spr=https&sr=b
&sv=2020-10-02





Describing Image...

{
    "description": "The image shows a laptop with a complaint message on
the screen, a smartphone, and a fan blowing air towards the laptop,
possibly related to electronic overheating issues.",
    "annotation": [
        {
            "bbox": [150, 200, 874, 700],
            "label": "laptop with complaint message"
        },
        {
            "bbox": [780, 550, 1000, 800],
            "label": "fan blowing air"
        },
        {
            "bbox": [650, 850, 900, 1000],
            "label": "smartphone"
        }
    ]
}
image description saved to :./output/image_description.txt
image annotation info saved to: ./output/image_description_annotation.json



Annotated Image Path: ./output/annotated_image.png
```

```
Classifying description...

Classification completed....
classification result saved to: ./output/classification.txt
The complaint related to a laptop potentially experiencing overheating
issues should be classified under:

- **Category**: Electronics
  - **Subcategory**: Computers & Tablets
```

Solution:

N/A