```
!ls
```

```
sample_data
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!unzip /content/drive/MyDrive/NPL_PROJECTS/roberta_base.zip -d roberta_base
```

```
Archive:  /content/drive/MyDrive/NPL_PROJECTS/roberta_base.zip
  inflating: roberta_base/tokenizer_config.json
  inflating: roberta_base/config.json
  inflating: roberta_base/special_tokens_map.json
  inflating: roberta_base/vocab.json
  inflating: roberta_base/merges.txt
  inflating: roberta_base/model.safetensors
  inflating: roberta_base/tokenizer.json
```

LABEL_0 → "Hate Speech" LABEL_1 → "Offensive". { predicted label } LABEL_2 → "Neither".

```
from transformers import pipeline

# Create a text-classification pipeline
classifier = pipeline("text-classification", model="/content/roberta_base", device=0)  # use device=-1 for CPU

# ---------------------------------------------------
# 3️⃣ Test your model
# ---------------------------------------------------
# Single test
print("Single example test:")
print(classifier("I hate you!"))

# Multiple examples
texts = [
    "hi! karan",
    "You're such a nice person.",
    "Go back to your country.",
    "i hate it!"
]

print("\nBatch test:")
for text in texts:
    pred = classifier(text)[0]
    print(f"Text: {text}")
    print(f"Predicted label: {pred['label']}, score: {pred['score']:.4f}")
    print("-" * 40)
```

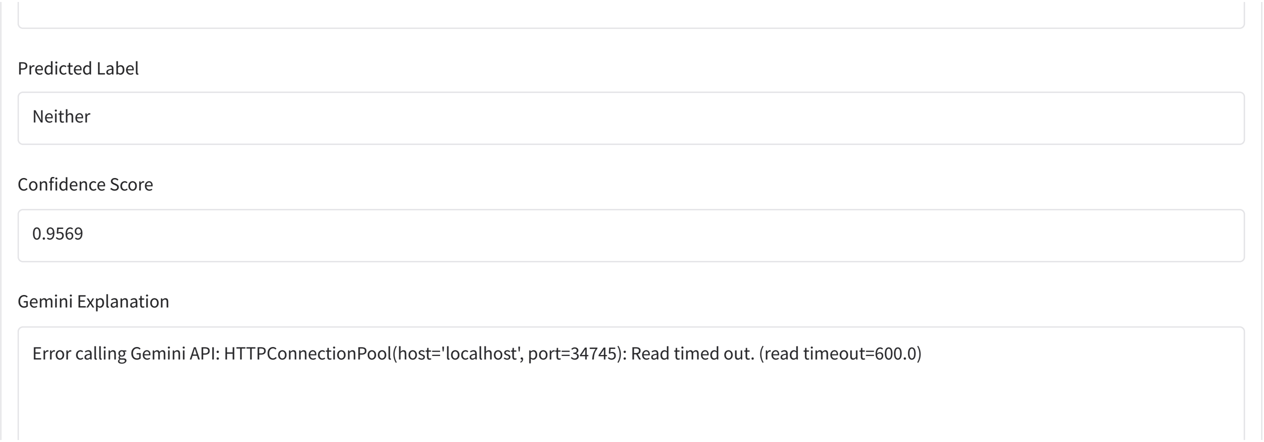
```
Device set to use cpu
Single example test:
[{'label': 'LABEL_1', 'score': 0.5674282908439636}]

Batch test:
Text: hi! karan
Predicted label: LABEL_2, score: 0.9842
----------------------------------------
Text: You're such a nice person.
Predicted label: LABEL_2, score: 0.8892
----------------------------------------
Text: Go back to your country.
Predicted label: LABEL_2, score: 0.9602
----------------------------------------
Text: i hate it!
Predicted label: LABEL_2, score: 0.7389
----------------------------------------
```

LABEL_0 → "Hate Speech" LABEL_1 → "Offensive". { predicted label } LABEL_2 → "Neither".

```
import google.generativeai as genai
from transformers import pipeline
import gradio as gr
```

```
# -----------------------
# 1. Configure Gemini
# -----------------------
api_key = "AIzaSyBR8XLAeY_69yHesF8NGhukHoMPVhsBYYI"   # apna Gemini API key yahan daalo
genai.configure(api_key=api_key)
gemini = genai.GenerativeModel("gemini-2.0-flash")  # ⚡ fast model

# -----------------------
# 2. Load classifier
# -----------------------
classifier = pipeline("text-classification", model="/content/roberta_base", device=0)

id2label = {
    "LABEL_0": "Hate Speech",
    "LABEL_1": "Offensive",
    "LABEL_2": "Neither"
}

# -----------------------
# 3. Streaming function
# -----------------------
def classify_and_explain(text):
    # Step 1: Classifier result (instant)
    pred_raw = classifier(text)[0]
    pred = id2label.get(pred_raw['label'], pred_raw['label'])
    score = f"{pred_raw['score']:.4f}"
    yield pred, score, "⏳ Gemini explanation loading..."

    # Step 2: Gemini explanation (slower)
    prompt = f'Text: "{text}"\nPrediction: {pred}\nExplain briefly in 1–2 sentences.'
    try:
        response = gemini.generate_content(prompt)
        yield pred, score, response.text.strip()
    except Exception as e:
        yield pred, score, f"Error calling Gemini API: {e}"

# -----------------------
# 4. Gradio Interface
# -----------------------
with gr.Blocks() as demo:
    gr.Markdown("## ⚡ Hate Speech Classifier + Gemini-2.0-Flash")
    gr.Markdown("Classifier runs instantly. Gemini explanation streams after.")

    text_input = gr.Textbox(label="Enter text", placeholder="Type a sentence...", lines=3)
    label_output = gr.Textbox(label="Predicted Label")
    score_output = gr.Textbox(label="Confidence Score")
    explanation_output = gr.Textbox(label="Gemini Explanation", lines=6)

    submit_btn = gr.Button("Classify & Explain")
    submit_btn.click(fn=classify_and_explain,
                     inputs=text_input,
                     outputs=[label_output, score_output, explanation_output],
                     show_progress="hidden")  # hides loading spinner

demo.launch()
```

```
Device set to use cpu
It looks like you are running Gradio on a hosted Jupyter notebook, which requires `share=True`. Automatically setting `s

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://8f85c33edff4f9afd2.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in
```



Predicted Label

Neither

Confidence Score

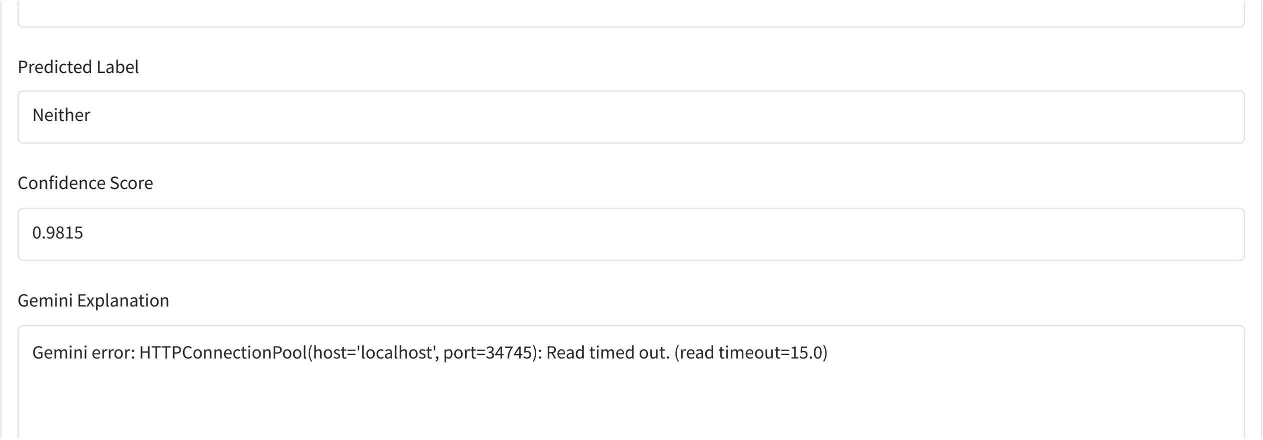
0.9569

Gemini Explanation

Error calling Gemini API: HTTPConnectionPool(host='localhost', port=34745): Read timed out. (read timeout=600.0)

```python
import google.generativeai as genai
from transformers import pipeline
import gradio as gr

# ------------------------
# 1. Configure Gemini
# ------------------------
api_key = "AIzaSyBR8XLAeY_69yHesF8NGhukHoMPVhsBYYI"   # apna Gemini API key
genai.configure(api_key=api_key)
gemini = genai.GenerativeModel("gemini-2.0-flash")  # ⚡ fast model

# ------------------------
# 2. Load classifier
# ------------------------
classifier = pipeline("text-classification", model="/content/roberta_base", device=0)

id2label = {
    "LABEL_0": "Hate Speech",
    "LABEL_1": "Offensive",
    "LABEL_2": "Neither"
}

# ------------------------
# 3. Function with timeout + error handling
# ------------------------
def classify_and_explain(text):
    # Step 1: Classifier result (instant)
    pred_raw = classifier(text)[0]
    pred = id2label.get(pred_raw['label'], pred_raw['label'])
    score = f"{pred_raw['score']:.4f}"
    yield pred, score, "⏳ Gemini explanation loading..."

    # Step 2: Gemini explanation (slower, max 15 sec)
    prompt = f'Text: "{text}"\nPrediction: {pred}\nExplain briefly in 1-2 sentences.'
    try:
        response = gemini.generate_content(
            prompt,
            request_options={"timeout": 15}   # ⏰ max wait 15 sec
        )
        explanation = response.text.strip()
    except Exception as e:
        explanation = f"Gemini error: {e}"

    yield pred, score, explanation

# ------------------------
# 4. Gradio Interface
# -----------------------
with gr.Blocks() as demo:
    gr.Markdown("## ⚡ Hate Speech Classifier + Gemini-2.0-Flash")
    gr.Markdown("Classifier runs instantly. Gemini explanation streams after.")

    text_input = gr.Textbox(label="Enter text", placeholder="Type a sentence...", lines=3)
    label_output = gr.Textbox(label="Predicted Label")
    score_output = gr.Textbox(label="Confidence Score")
    explanation_output = gr.Textbox(label="Gemini Explanation", lines=6)

    submit_btn = gr.Button("Classify & Explain")
    submit_btn.click(
        fn=classify_and_explain,
        inputs=text_input,
        outputs=[label_output, score_output, explanation_output],
        show_progress="hidden"
    )

demo.launch()
```

```
Device set to use cpu
It looks like you are running Gradio on a hosted Jupyter notebook, which requires `share=True`. Automatically setting `s

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://88a153c628cccdfd25.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in
```



Predicted Label

Neither

Confidence Score

0.9815

Gemini Explanation

Gemini error: HTTPConnectionPool(host='localhost', port=34745): Read timed out. (read timeout=15.0)

```
import google.generativeai as genai
from transformers import pipeline

# -----------------------
# 1. Configure Gemini
# -----------------------
api_key = "AIzaSyBR8XLAeY_69yHesF8NGhukHoMPVhsBYYI"   # apna API key
genai.configure(api_key=api_key)
gemini = genai.GenerativeModel("gemini-2.0-flash")  # fast model

# -----------------------
# 2. Load classifier
# -----------------------
classifier = pipeline("text-classification", model="/content/roberta_base", device=0)

id2label = {
    "LABEL_0": "Hate Speech",
    "LABEL_1": "Offensive",
    "LABEL_2": "Neither"
}

# -----------------------
# 3. Integration Function
# -----------------------
```

```
def classify_and_explain(text):
    # Step 1: Classifier prediction
    pred_raw = classifier(text)[0]
    pred = id2label.get(pred_raw['label'], pred_raw['label'])
    score = f"{pred_raw['score']:.4f}"

    # Step 2: LLM Explanation (with fallback)
    prompt = f"""
    You are an expert NLP assistant.
    Text: "{text}"
    Classifier predicted: {pred} (confidence {score}).

    ✅ Task: Briefly explain why this prediction makes sense
    OR point out if the classifier might be wrong.
    Keep it concise (2–3 sentences).
    """
    try:
        response = gemini.generate_content(prompt, request_options={"timeout": 60})
        explanation = response.text.strip()
    except Exception:
        explanation = "⚠️ Gemini timed out. Showing only classifier result."

    return pred, score, explanation

# ------------------------
# 4. Example usage
# ------------------------
txt = "I hate people like you."
label, conf, exp = classify_and_explain(txt)

print("Predicted Label:", label)
print("Confidence:", conf)
print("Explanation:", exp)
```

```
Device set to use cpu
Predicted Label: Offensive
Confidence: 0.5050
Explanation: ⚠️ Gemini timed out. Showing only classifier result.
```

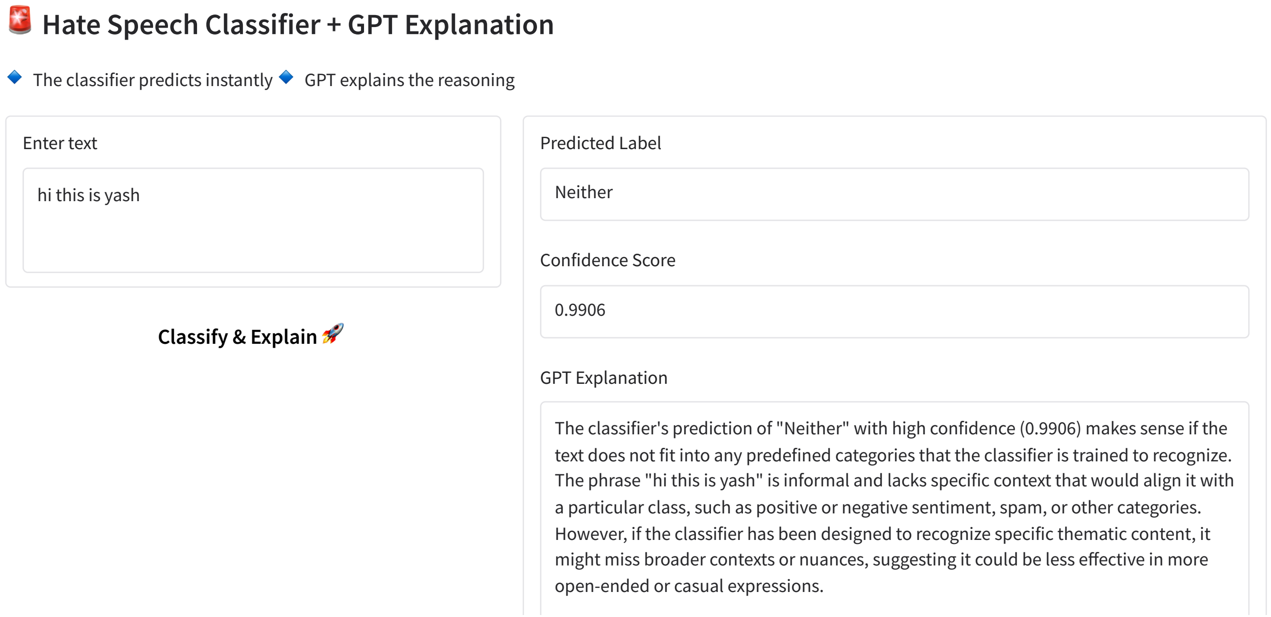
```
from openai import OpenAI
from transformers import pipeline

# 1. Configure OpenAI
client = OpenAI(api_key="sk-proj-BMLPuGeRnxZALkxx_TJ7eky4SFHTaGcyOfTs-F_18YXRgOO9-vD2km2RVOWJYRK-hGVBZ8FXzcT3BlbkFJNR7pW20qml

# 2. Load classifier
classifier = pipeline("text-classification", model="/content/roberta_base", device=0)

id2label = {
    "LABEL_0": "Hate Speech",
    "LABEL_1": "Offensive",
    "LABEL_2": "Neither"
}

# 3. Integration function
def classify_and_explain(text):
    # Classifier output
    pred_raw = classifier(text)[0]
    pred = id2label.get(pred_raw['label'], pred_raw['label'])
    score = f"{pred_raw['score']:.4f}"

    # Ask LLM for explanation
    prompt = f"""
    Text: "{text}"
    Classifier prediction: {pred} (confidence {score}).

    Briefly explain why this makes sense,
    or suggest if the classifier might be wrong (2–3 sentences).
    """

    try:
        response = client.chat.completions.create(
            model="gpt-4o-mini",  # fast & cheap
            messages=[{"role": "user", "content": prompt}],
            timeout=15
        )
        explanation = response.choices[0].message.content.strip()
    except Exception as e:
        explanation = f"⚠️ OpenAI error: {e}"

    return pred, score, explanation
```

```
# Test
txt = "I hate people like you."
label, conf, exp = classify_and_explain(txt)
print(label, conf, exp)
```

```
Device set to use cpu
Offensive 0.5050 The classifier's prediction of "Offensive" with a confidence of 0.5050 makes sense given that the phras
```

```
from openai import OpenAI
from transformers import pipeline
import gradio as gr

# -----------------------
# 1. Configure OpenAI
# -----------------------
client = OpenAI(api_key="sk-proj-BMLPuGeRnxZALkxx_TJ7eky4SFHTaGcyOfTs-F_18YXRgOO9-vD2km2RVOWJYRK-hGVBZ8FXzcT3BlbkFJNR7pW20qml

# -----------------------
# 2. Load classifier
# -----------------------
classifier = pipeline("text-classification", model="/content/roberta_base", device=0)

id2label = {
    "LABEL_0": "Hate Speech",
    "LABEL_1": "Offensive",
    "LABEL_2": "Neither"
}

# -----------------------
# 3. Integration function
# -----------------------
def classify_and_explain(text):
    if not text.strip():
        return "⚠️ Please enter some text", "", ""

    # Step 1: Classifier result
    pred_raw = classifier(text)[0]
    pred = id2label.get(pred_raw['label'], pred_raw['label'])
    score = f"{pred_raw['score']:.4f}"

    # Step 2: Ask LLM for explanation
    prompt = f"""
    Text: "{text}"
    Classifier prediction: {pred} (confidence {score}).

    Briefly explain why this makes sense,
    or suggest if the classifier might be wrong (2–3 sentences).
    """

    try:
        response = client.chat.completions.create(
            model="gpt-4o-mini",  # fast + cheap
            messages=[{"role": "user", "content": prompt}],
            timeout=15
        )
        explanation = response.choices[0].message.content.strip()
    except Exception as e:
        explanation = f"⚠️ OpenAI error: {e}"

    return pred, score, explanation

# -----------------------
# 4. Gradio Interface
# -----------------------
with gr.Blocks() as demo:
    gr.Markdown("## 🚨 Hate Speech Classifier + GPT Explanation")
    gr.Markdown("🔹 The classifier predicts instantly\n🔹 GPT explains the reasoning")

    with gr.Row():
        with gr.Column(scale=2):
            text_input = gr.Textbox(
                label="Enter text",
                placeholder="Type a sentence to classify...",
                lines=3
            )
            submit_btn = gr.Button("Classify & Explain 🚀")

        with gr.Column(scale=3):
            label_output = gr.Textbox(label="Predicted Label")
            score_output = gr.Textbox(label="Confidence Score")
            explanation_output = gr.Textbox(label="GPT Explanation", lines=6)

    submit_btn.click(
        fn=classify_and_explain,
        inputs=text_input,
        outputs=[label_output, score_output, explanation_output]
    )

demo.launch()
```

```
Device set to use cpu
It looks like you are running Gradio on a hosted Jupyter notebook, which requires `share=True`. Automatically setting `s

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://f0b0ce25b47856970c.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in
```



🚨 Hate Speech Classifier + GPT Explanation

🔷 The classifier predicts instantly 🔷 GPT explains the reasoning

Enter text

hi this is yash

Classify & Explain 🚀

Predicted Label

Neither

Confidence Score

0.9906

GPT Explanation

The classifier's prediction of "Neither" with high confidence (0.9906) makes sense if the text does not fit into any predefined categories that the classifier is trained to recognize. The phrase "hi this is yash" is informal and lacks specific context that would align it with a particular class, such as positive or negative sentiment, spam, or other categories. However, if the classifier has been designed to recognize specific thematic content, it might miss broader contexts or nuances, suggesting it could be less effective in more open-ended or casual expressions.