Lab Work: Text Summarization Using Hugging Face Transformers

Estimated Time: 60 minutes
Prerequisites: Basic knowledge of Python, Flask, and REST APIs.

Lab Overview

In this lab, you will create a text summarization application using Hugging Face's free transformer models. The application will take long text input and generate concise summaries. You'll then deploy it as a web application using Flask. By the end, you'll have a functional web application for text summarization.

Lab Objectives

By the end of this lab, you will be able to:

1. Set up a Python environment with required libraries
2. Create a text summarization function using Hugging Face
3. Package the application for reuse
4. Deploy the application using Flask
5. Implement error handling and basic testing

Lab Setup

Required Libraries

Install these packages before starting:

```
pip install transformers flask torch
```

Lab Tasks

Task 1: Create Text Summarization Function

Objective: Develop a Python function for text summarization.

1. Create text_summarizer.py:

```
from transformers import pipeline
```

```
def summarize_text(text_to_summarize):
    try:
        # Load summarization pipeline with a small, free model
        summarizer = pipeline("summarization", model="facebook/bart-large-cnn"
)

        # Generate summary (limiting input length for free tier)
        if len(text_to_summarize) > 1024:
            text_to_summarize = text_to_summarize[:1024]

        summary = summarizer(text_to_summarize, max_length=130, min_length
=30, do_sample=False)
        return summary[0]['summary_text']
    except Exception as e:
        print(f"Error during summarization: {e}")
        return None
```

2. Test the function:

```
from text_summarizer import summarize_text
sample_text = "The quick brown fox jumps over the lazy dog. This sentence co
ntains all the letters in the English alphabet. It is often used for typing practice."
print(summarize_text(sample_text))
```

Task 2: Create Flask Web Application

Objective: Build a web interface for the summarizer.

1. Create app.py:

```
from flask import Flask, render_template, request
from text_summarizer import summarize_text

app = Flask("Text Summarizer")

@app.route("/")
def home():
    return render_template('index.html')

@app.route("/summarize", methods=['POST'])
def summarize():
    if request.method == 'POST':
        text = request.form['text_to_summarize']
        if not text.strip():
```

```
        return "Please enter some text to summarize."

    summary = summarize_text(text)
    if summary:
        return f"<h2>Summary:</h2><p>{summary}</p>"
    else:
        return "Error generating summary. Please try again."

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

2. Create templates/index.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Text Summarizer</title>
    <style>
        body { font-family: Arial, sans-serif; max-width: 800px; margin: 0 auto; pa
dding: 20px; }
        textarea { width: 100%; height: 200px; margin-bottom: 10px; }
        button { padding: 10px 15px; background-color: #4CAF50; color: white; b
order: none; cursor: pointer; }
    </style>
</head>
<body>
    <h1>Text Summarizer</h1>
    <form action="/summarize" method="post">
        <textarea name="text_to_summarize" placeholder="Enter text to summariz
e..." required></textarea>
        <button type="submit">Summarize Text</button>
    </form>
</body>
</html>
```

Task 3: Run and Test the Application

Objective: Launch and test your web application.

1. Run the application:

```
python app.py
```

2. Open a web browser and navigate to:

```
http://localhost:5000
```

3. Test with sample text:

```
The field of artificial intelligence has grown significantly in recent years. Machine learning, a subset of AI, involves algorithms that improve automatically through experience. Deep learning uses neural networks with many layers. These technologies power applications like speech recognition, image classification, and autonomous vehicles. AI is transforming industries from healthcare to finance, though ethical concerns remain about bias and job displacement.
```

Expected Output: A concise summary of the input text.

Task 4: Add Error Handling

Objective: Improve robustness with better error handling.

Modify text_summarizer.py:

```python
def summarize_text(text_to_summarize):
    try:
        if not text_to_summarize or not isinstance(text_to_summarize, str):
            raise ValueError("Invalid input text")

        if len(text_to_summarize) < 30:
            raise ValueError("Text too short to summarize")

        summarizer = pipeline("summarization", model="facebook/bart-large-cnn")

        if len(text_to_summarize) > 1024:
            text_to_summarize = text_to_summarize[:1024]

        summary = summarizer(text_to_summarize, max_length=130, min_length=30, do_sample=False)
        return summary[0]['summary_text']

    except ValueError as ve:
        print(f"Input validation error: {ve}")
        return None
    except Exception as e:
        print(f"Summarization error: {e}")
        return None
```

Task 5: Package the Application

Objective: Organize your code into a proper package structure.

1. Create this directory structure:

```
text_summarizer/
│    ├──── __init__.py
│    ├──── summarizer.py
│    └──── templates/
│         └──── index.html
```

2. Move the respective files to their new locations
3. Update import statements accordingly

Optional Enhancements:

1. Add length limits and character counters to the web interface
2. Implement loading indicators for longer texts
3. Add support for file uploads (txt, pdf)
4. Deploy to a free cloud service like Render or Railway