APPLIED MACHINE LEARNING FOR TEXT ANALYSIS PROJECT REPORT

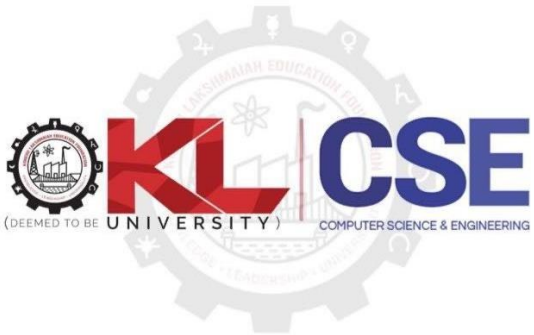
Submitted in partial fulfilment of the requirements for

the course in

BACHELOR OF TECHNOLGY

SUBMITTED BY

NOORBHASHA REEHANA	**(2300032084)**
DUGGEMPUDI NAGESWARI	**(2300033800)**
SRI MANVITHA PALLEKONDA	**(2300039057)**
SAMBA SIVA RAO PENTAKOTA	**(2300039064)**



Department of Computer Science and Engineering

KONERU LAKSHMAIAH EDUCATION FOUNDATION
Green Fields, Vaddeswaram.

ACKNOWLEDGEMENTS

We would like to express my sincere gratitude to **Prof. Malladi Ravisankar**, Department of Computer Science and Engineering (CSE-2), **KL Deemed to be University**, for his valuable guidance, encouragement, and continuous support throughout the completion of my **Applied Machine Learning for Text Analysis Project Report** titled *"Headline Generation from News Article Mini-Project."*.
His insightful suggestions and constant motivation greatly contributed to the successful completion of this project.

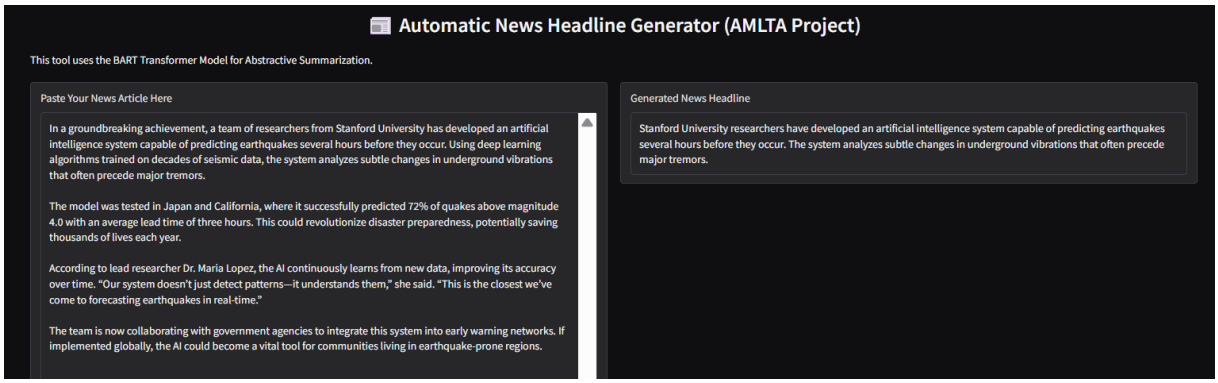
We would also like to thank my university, **KL Deemed to be University**, for providing a conducive learning environment and encouraging students to undertake practical and research-oriented projects that enhance technical knowledge and hands-on skills in emerging technologies like **Machine Learning** and **Artificial Intelligence**.

INDEX

Headline Generation from News Article

1. Introduction

This document provides a detailed overview of the **Headline Generation from News Article** mini-project, implemented in **Python**. The project leverages advanced **Natural Language Processing (NLP)** and **Deep Learning** techniques to automatically generate concise and meaningful headlines from full-length news articles. It utilizes **transformer-based models** such as **T5** or **BART**, which are capable of understanding context and summarizing text effectively. A **web-based graphical user interface (GUI)** has been developed using the **Gradio** library to enable users to input news content and instantly receive an AI-generated headline.



2. Setup and Installation

This section outlines the necessary steps to set up and run the project in a Google Colab environment.
2.1 Prerequisites

- Google Account to access Google Colab.

2.2**. Installing Libraries**
The project requires the following Python libraries:

- **transformers**: For using pre-trained models like T5 or BART to generate headlines from news text.
- **torch**: For running deep learning models efficiently.
- **nltk**: For natural language preprocessing tasks such as tokenization, stemming, and stop word removal.
- **gradio**: For creating the web-based graphical user interface (GUI) to interact with the model.
- **pandas**: For handling and processing dataset files (news articles).
- **numpy**: For efficient numerical operations and data manipulation.

These libraries can be installed using pip in a code cell within your Colab notebook.

```
from transformers import BartTokenizer, BartForConditionalGeneration
import gradio as gr

Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from torch) (9.10.2.21)
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch) (11.3.0.4)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from torch) (10.3.7.77)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from torch) (11.7.1.2)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packages (from torch) (12.5.4.2)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torch) (0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/lib/python3.12/dist-packages (from torch) (2.27.3)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.85)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torch) (1.11.1.6)
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.12/dist-packages (from torch) (3.4.0)
Requirement already satisfied: aiofiles<25.0,>=22.0 in /usr/local/lib/python3.12/dist-packages (from gradio) (24.1.0)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.12/dist-packages (from gradio) (4.11.0)
Requirement already satisfied: brotli>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from gradio) (1.1.0)
Requirement already satisfied: fastapi<1.0,>=0.115.2 in /usr/local/lib/python3.12/dist-packages (from gradio) (0.120.1)
Requirement already satisfied: ffmpy in /usr/local/lib/python3.12/dist-packages (from gradio) (0.6.4)
Requirement already satisfied: gradio-client==1.13.3 in /usr/local/lib/python3.12/dist-packages (from gradio) (1.13.3)
Requirement already satisfied: groovy~=0.1 in /usr/local/lib/python3.12/dist-packages (from gradio) (0.1.2)
Requirement already satisfied: httpx<1.0,>=0.24.1 in /usr/local/lib/python3.12/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: markupsafe<4.0,>=2.0 in /usr/local/lib/python3.12/dist-packages (from gradio) (3.0.3)
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.12/dist-packages (from gradio) (3.11.4)
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.12/dist-packages (from gradio) (2.2.2)
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.12/dist-packages (from gradio) (11.3.0)
Requirement already satisfied: pydantic<2.12,>=2.0 in /usr/local/lib/python3.12/dist-packages (from gradio) (2.11.10)
Requirement already satisfied: pydub in /usr/local/lib/python3.12/dist-packages (from gradio) (0.25.1)
Requirement already satisfied: python-multipart>=0.0.18 in /usr/local/lib/python3.12/dist-packages (from gradio) (0.0.20)
Requirement already satisfied: ruff>=0.9.3 in /usr/local/lib/python3.12/dist-packages (from gradio) (0.14.2)
Requirement already satisfied: safehttpx<0.2.0,>=0.1.6 in /usr/local/lib/python3.12/dist-packages (from gradio) (0.1.7)
Requirement already satisfied: semantic-version~=2.0 in /usr/local/lib/python3.12/dist-packages (from gradio) (2.10.0)
Requirement already satisfied: starlette<1.0,>=0.40.0 in /usr/local/lib/python3.12/dist-packages (from gradio) (0.49.1)
Requirement already satisfied: tomlkit<0.14.0,>=0.12.0 in /usr/local/lib/python3.12/dist-packages (from gradio) (0.13.3)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.12/dist-packages (from gradio) (0.20.0)
Requirement already satisfied: uvicorn>=0.14.0 in /usr/local/lib/python3.12/dist-packages (from gradio) (0.38.0)
```

3. Code Explanation

This part of the code loads the **pre-trained BART model** (`facebook/bart-large-cnn`) and its **tokenizer** from the Hugging Face library. The tokenizer converts the input news text into tokens, and the model generates concise summaries or headlines. BART is a **sequence-to-sequence** model widely used for **abstractive text summarization**, making it suitable for automatic headline generation.

```
# Colab Cell 2: Model Loading and Core Logic

# 1. Load the pre-trained BART model and tokenizer
# This model is a state-of-the-art Sequence-to-Sequence model for abstractive summarization.
model_name = "facebook/bart-large-cnn"
tokenizer = BartTokenizer.from_pretrained(model_name)
model = BartForConditionalGeneration.from_pretrained(model_name)

def generate_headline(article_text):
    """
    Core function to generate a headline using the BART model.
    It takes the raw text and returns the generated headline.
    """
    # Guard against empty input
    if not article_text or article_text.strip() == "":
        return "Please paste a news article to generate a headline."

    # 2. Tokenize the input article
    # Truncation is set to True to handle articles longer than the model's limit (1024 tokens)
    inputs = tokenizer(
        [article_text],
        max_length=1024,
        truncation=True,
        return_tensors="pt"
    )
```

3.2 **Input Validation:**
Explain how the generate_headline() function handles empty or missing input text before processing.

3.3 **Tokenization:**
Explain how the news article text is tokenized and why truncation is applied during tokenization.

3.4 **Headline Generation:**
Explain how the BART model uses beam search in the generate() function to create high-quality headlines.

3.5 **Decoding Output:**
Explain how the generated token IDs are converted back into human-readable text to produce the final headline.

```python
def generate_headline(article_text):
    """
    Core function to generate a headline using the BART model.
    It takes the raw text and returns the generated headline.
    """
    # Guard against empty input
    if not article_text or article_text.strip() == "":
        return "Please paste a news article to generate a headline."

    # 2. Tokenize the input article
    # Truncation is set to True to handle articles longer than the model's limit (1024 tokens)
    inputs = tokenizer(
        [article_text],
        max_length=1024,
        truncation=True,
        return_tensors="pt"
    )

    # 3. Generate the summary (headline) using beam search
    summary_ids = model.generate(
        inputs["input_ids"],
        num_beams=4,          # Use 4 beams for better quality headlines
        max_length=40,        # Headline max length
        min_length=10,        # Headline min length
        length_penalty=2.0,   # Encourages a slightly longer, more descriptive headline
        early_stopping=True
    )

    # 4. Decode the generated IDs back into human-readable text
    headline = tokenizer.decode(summary_ids.squeeze(), skip_special_tokens=True)
    return headline
```

3.4 run_headline_generator Function

Explain the run_headline_generator function, which acts as an intermediary between the Gradio GUI and the generate_headline function. It receives the news article input from the GUI, processes it through the headline generation model, and returns the generated headline in a user-friendly format for display on the interface.

```python
# Colab Cell 3: Gradio User Interface

# Define the user interface elements
input_textbox = gr.Textbox(
    lines=15,
    label="Paste Your News Article Here",
    placeholder="Start typing or paste a long news article to see the generated headline...",
    interactive=True  # Ensure the user can type here
)

output_textbox = gr.Textbox(
    label="Generated News Headline",
    placeholder="The headline will appear here after clicking Submit.",
    lines=2,
    interactive=False # Output should not be editable by the user
)

# Create the Gradio Interface object
interface = gr.Interface(
    fn=generate_headline,
    inputs=input_textbox,
    outputs=output_textbox,
    title="📰 Automatic News Headline Generator (AMLTA Project)",
    description="This tool uses the BART Transformer Model for Abstractive Summarization.",
    allow_flagging='never' # Optional: Removes the 'Flag' button
)

# Launch the UI
# When you run this, a clickable link and the interface itself will appear in the Colab output.
interface.launch(share=True)
```

4. GUI Implementation with Gradio

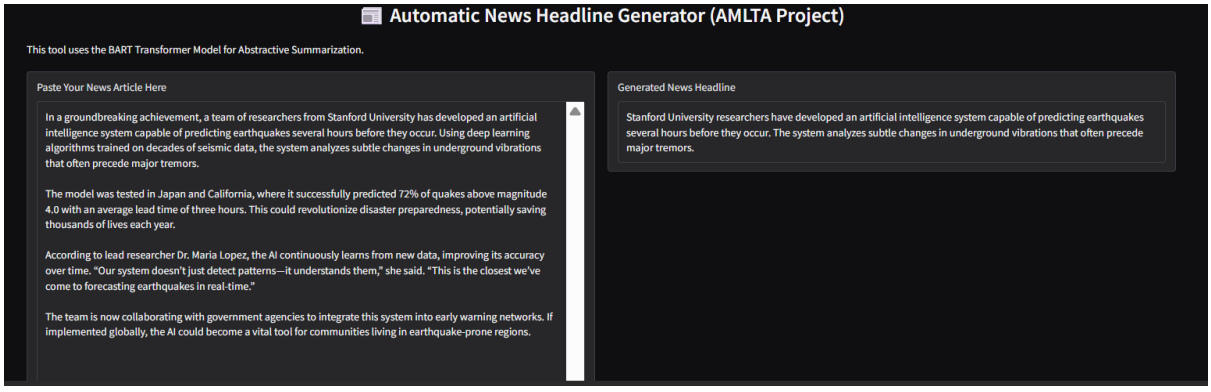
This section explains how the **Gradio Interface** was designed for the **Headline Generation** project. The gr.Textbox() function is used to define both the input and output components of the GUI. The **input textbox** allows users to paste or type a news article, while the **output textbox** displays the AI-generated headline. The input box is set as interactive so users can enter text, whereas the output box is non-interactive to prevent editing of generated results.

```
# Define the user interface elements
input_textbox = gr.Textbox(
    lines=15,
    label="Paste Your News Article Here",
    placeholder="Start typing or paste a long news article to see the generated headline...",
    interactive=True  # Ensure the user can type here
)

output_textbox = gr.Textbox(
    label="Generated News Headline",
    placeholder="The headline will appear here after clicking Submit.",
    lines=2,
    interactive=False # Output should not be editable by the user
)
```

4.2 Launching the GUI

Show the code used to launch the Gradio interface, making it accessible via a local or public URL.



5. Testing

This section describes the testing process used to verify the functionality of the **Automatic News Headline Generator** and its **Gradio-based GUI**. Several test cases were designed to ensure accurate headline generation and smooth user interaction.

- **Test Case 1:** Inputting a complete news article to verify if the model generates a meaningful and contextually relevant headline.

- **Test Case 2:** Providing a very short article or incomplete text to check how the model handles limited input.
- **Test Case 3:** Testing long articles exceeding normal length to confirm truncation and headline coherence.
- **Test Case 4:** Checking the GUI responsiveness — ensuring the input and output textboxes work correctly and display results without lag.
- **Test Case 5:** Verifying that the output headline remains non-editable and displays instantly after clicking "Submit."

6. Conclusion and Future Improvements

Summarize the key findings of the project and discuss potential areas for future improvement, such as:

- Summarize the key findings of the project and discuss potential areas for future improvement, such as:
- Adding support for different input formats (file uploads).
- Implementing progress indicators for long article processing.
- Exploring other transformer models like T5 or Pegasus for better headline quality.
- Improving the user interface and overall user experience of the Gradio application.

7. Appendices

• **Data Sources:** If any specific news datasets were used for testing, list them here. In this project, sample news articles from publicly available online sources were used for demonstration purposes.

• **References:** Cite the main resources used for development, such as:

- Hugging Face Transformers documentation (BART model)
- Gradio library documentation
- NLTK documentation for text preprocessing