NLP Report
Topic: Talk to a President
Disha Kacha (G21218299)

Index Table:

Table of figures:

Introduction:

This project focuses on developing a chatbot using a dataset of speeches by U.S. and Russian presidents. The primary goal was to fine-tune a Retrieval-Augmented Generation (RAG) model to enable the chatbot to respond accurately to user queries based on the dataset and simulate responses in the style of a president. The U.S. speech dataset was already clean and required no additional preprocessing. However, the Russian dataset needed significant cleaning. During the merging process, some columns were removed, and the *transcript* column was identified as the main source for generating the data since it contained the full text of the speeches.

Problem Statement:
The project addresses challenges in analyzing presidential speeches, including inconsistent data quality, inefficient retrieval, limited contextual understanding, high computational demands, and lack of fine-tuning for task-specific needs. Existing models like FAISS faced compatibility issues, while traditional sequence models like LSTMs struggled with context, and transformer-based models required significant computational resources. The goal is to develop an NLP-powered chatbot that retrieves relevant excerpts, summarizes content, performs sentiment analysis, and simulates responses in a presidential style, ensuring compatibility, scalability, and user-friendly integration.

Method 1 using RAG Faiss:

To begin the project, I explored various approaches to understand how to implement a chatbot capable of generating relevant responses based on presidential speeches. Initially, I experimented with the code provided below, which was generated using ChatGPT. This code utilized FAISS from the Hugging Face library for document retrieval, combined with sequence generation using the BART model.
The main objective of this phase was to learn how a Retrieval-Augmented Generation (RAG) model works and how components like tokenizers, retrievers, and generators interact. However, during execution, I encountered significant challenges. Specifically, the FAISS library required a Conda environment, which was incompatible with my current instance setup. This resulted in multiple errors and made it difficult to proceed with this approach.

```
# Function to create FAISS index for document embeddings
def create_faiss_index(document_embeddings):
    try:
        # Convert the embeddings to float32 (ensure proper format for FAISS)
        document_embeddings = np.array(document_embeddings).astype('float32')

        # Normalize embeddings before indexing (FAISS requires normalized embeddings for L2 search)
        faiss.normalize_L2(document_embeddings)

        # Initialize FAISS index (IndexFlatL2 for L2 distance search)
        index = faiss.IndexFlatL2(document_embeddings.shape[1])

        # Add document embeddings to the FAISS index
        index.add(document_embeddings)

        return index
    except Exception as e:
        raise ValueError(f"Error creating FAISS index: {str(e)}")

class FAISSRetriever:
    def __init__(self, faiss_index, embeddings_model):
        self.index = faiss_index
        self.embeddings_model = embeddings_model

    def retrieve(self, query, top_k=1):
        query_embedding = np.array(self.embeddings_model.embed([query])).astype('float32')

        D, I = self.index.search(query_embedding, top_k)

        return I
```

Fig.1: RAG using FAISS approach

This experience prompted me to take a step back, analyze the limitations, and begin crafting my own logic for the model, ensuring compatibility with my setup while tailoring it to meet the project requirements.

Method 2 using Cosine Similarity:

After encountering challenges with the FAISS-based approach, I decided to develop a simpler and more compatible method for retrieving relevant speeches. The main objective was to create a solution that aligns with my existing environment and effectively identifies speeches that match user queries. To achieve this, I implemented a retrieval mechanism using TF-IDF vectorization and cosine similarity.
This approach offers several advantages:

1. Environment Compatibility: Unlike FAISS, which had dependencies incompatible with my setup, TF-IDF and cosine similarity work seamlessly with the libraries already available in my environment.
2. Simplicity and Interpretability: This method is easy to understand, implement, and debug, making it an ideal starting point for exploring speech similarity.
3. Effectiveness for Text-Based Data: TF-IDF captures the importance of terms in a document relative to the entire dataset, and cosine similarity effectively measures textual similarity between the query and speeches.

```
def get_most_similar_speech(query, metadata_df):
    speeches = metadata_df['transcript'].tolist()
    vectorizer = TfidfVectorizer(stop_words='english')
    tfidf_matrix = vectorizer.fit_transform(speeches)
    query_vector = vectorizer.transform([query])
    cosine_similarities = cosine_similarity(query_vector, tfidf_matrix)
    most_similar_idx = cosine_similarities.argmax()
    similar_speech = metadata_df.iloc[most_similar_idx]
    return similar_speech, cosine_similarities[0][most_similar_idx]
```

Fig.2: Cosine Approach to check similarity

```
So, with purpose and resolve we turn to the tasks of our time.

Sustained by faith.

Driven by conviction.

And, devoted to one another and to this country we love with all our hearts.

May God bless America and may God protect our troops.

Thank you, America.

Similarity Score: 0.09067917067806751
```

Fig.3: Some part of Output after cosine

The similarity score, which in this case was approximately 0.0907, represents the cosine similarity between the query and the most similar speech in the dataset. This value is a measure of how closely the query aligns with the content of the retrieved speech, with a range from 0 to 1.

This implementation was developed by me as a basic understanding exercise to explore the functionality of retrieval systems and to build a strong foundation for the project. By building and testing this custom implementation, I gained a deeper understanding of retrieval systems, enabling me to better tailor the solution to the project's goals. Additionally, this experience reinforced the importance of adapting methods to the constraints of the environment and the dataset.

Method 3 using LSTM:

In an attempt to explore advanced techniques for text-based retrieval and generation, I decided to implement an LSTM-based Seq2Seq model. I wrote the LSTM function on my own, while some other parts of the code were generated with assistance from ChatGPT. This hybrid approach was intended to deepen my understanding of neural network architectures and their application to text generation tasks.

The implementation involved:

- Data Preprocessing: Tokenizing sentences and creating a vocabulary to convert words into indices.
- LSTM Seq2Seq Model: A custom sequence-to-sequence model with separate encoder and decoder LSTMs, embedding layers, and fully connected layers for predicting the next word in a sequence.
- Retrieval Integration: Using TF-IDF and cosine similarity to retrieve the top-N documents relevant to a user query.

```
class Seq2Seq(nn.Module):
    def __init__(self, vocab_size, hidden_dim, embedding_dim):
        super(Seq2Seq, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx=0)
        self.encoder_lstm = nn.LSTM(embedding_dim, hidden_dim, batch_first=True)
        self.decoder_lstm = nn.LSTM(embedding_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, vocab_size)

    def forward(self, input_tensor, hidden):
        embedded = self.embedding(input_tensor)
        encoder_outputs, (hidden, cell) = self.encoder_lstm(embedded, hidden)
        return encoder_outputs, (hidden, cell)
```

Fig.4: LSTM

The LSTM hidden and cell states were improperly initialized. Since my input tensor was unbatched (2-D), the hidden and cell states needed to be 2-D tensors as well. Initially, I attempted using 3-D tensors, which led to the runtime error. There were inconsistencies in how the batch dimension was handled during training and generation, resulting in misaligned tensor shapes. Based on my research, LSTMs, while effective for sequence tasks, may not be the best fit for retrieval-based models. More modern transformer-based models like BERT or GPT are often better suited for these use cases, as they capture context more effectively.

Method 4 using BERT/BART:

To enhance the retrieval and summarization process, I explored a solution combining Sentence-BERT (for document similarity and retrieval) and BART (for question-answering and text generation). This approach aimed to leverage advanced transformer models for better semantic understanding and answer generation.
The code was implemented with the following steps:

1. Document Embedding with Sentence-BERT:
    - Sentence-BERT (MiniLM-L6-v2) was used to encode each document from the dataset into dense embeddings. This allowed for efficient semantic similarity calculations using cosine similarity.
    - Precomputing embeddings reduced computation time for each query, making the system more efficient.
2. Document Retrieval:
    - For a given query, its embedding was computed using the same Sentence-BERT model.
    - Cosine similarity scores between the query embedding and precomputed document embeddings determined the top N relevant documents.
    - The logic for sorting and selecting top documents was customized, ensuring high-scoring documents were prioritized.

3. Answer Generation with BART:
    - BART (a transformer-based text generation model) was used to generate answers based on a combined context of the top retrieved documents.
    - To ensure relevance, parameters like num_beams, no_repeat_ngram_size, and max_length were tuned.
4. Custom Modifications:
    - I introduced specific logic to refine retrieval and generation, such as combining retrieved documents into a single context and increasing the maximum output length for more comprehensive answers.

```python
def retrieve_documents_with_embeddings(query, documents, top_n=5):
    query_embedding = sentence_model.encode([query])
    cosine_similarities = cosine_similarity(query_embedding, document_embeddings).flatten()
    top_n_indices = np.argsort(cosine_similarities)[-top_n:][::-1]
    retrieved_docs = [documents[i] for i in top_n_indices]
    return retrieved_docs, cosine_similarities[top_n_indices]
1 usage
def generate_answer(question, context):
    input_text = f"Answer the question based on the context provided: question: {question} context:
    print("Input text: ", input_text)
    inputs = bart_tokenizer.encode(input_text, return_tensors="pt", max_length=1024, truncation=Tru

    output = bart_model.generate(inputs,
                                 max_length=1000,
                                 num_beams=5,
                                 no_repeat_ngram_size=3,
                                 early_stopping=True)
    answer = bart_tokenizer.decode(output[0], skip_special_tokens=True)
    return answer
```

Fig.5: Retrieve and Generate Answer

```
Enter a question: What is the main conern related to climate change?
What is the main conern related to climate change?

Retrieved Documents:
Doc 1 (Score: 0.3081): Thank you.

Mr. President, Mr. Secretary-General, my fellow leaders, in the last year, our world has experienced great upheaval: a growing crisis in food insecurity; record h

Let us speak plainly. A permanent member of the United Nations Security Council invaded its neighbor, attempted to erase a sovereign state from the map.

Russia has shamelessly violated the core tenets of the United Nations Charter—no more important than the clear prohibition against countries taking the territory

Again, just today, President Putin has made overt nuclear threats against Europe and a reckless disregard for the responsibilities of the non-proliferation regime

Now Russia is calling—calling up more soldiers to join the fight. And the Kremlin is organizing a sham referenda to try to annex parts of Ukraine, an extremely si
Doc 2 (Score: 0.2962): My fellow Americans:

There's an old saying we've all heard a thousand times about the weather and how everyone talks about it but no one does anything about it. Well, many of you must
```

Fig.6: BERT/BART Output

The implementation faced several challenges. One significant issue was over-inclusiveness in retrieval, where the system often returned entire speeches, even when only parts of the text were relevant to the query. This occasionally resulted in the inclusion of unrelated sections, which diluted the relevance of the combined context used for generating answers. Additionally, the BART model exhibited limitations by generating responses based on the full context, which sometimes led to redundant or overly general information. Without fine-tuning on a task-specific dataset, the generated answers occasionally lacked precision and coherence. Moreover, the complexity and efficiency of the process posed challenges, as

combining multiple large documents into a single context for BART increased memory and computational demands, particularly with larger datasets. Although precomputing embeddings improved efficiency, the overall pipeline required further optimization for scalability. Despite these challenges, the implementation was a valuable step toward building a robust retrieval and generation system. Combining BERT for semantic similarity and BART for answer generation yielded promising results, but the experience underscored the need for fine-tuning models to align better with the dataset and task objectives, refining retrieval logic to focus on the most relevant sections of documents, and exploring alternative models or techniques for more precise and efficient answer generation.

My Contribution in Project:

My primary contribution to the project centered around implementing and refining the BART model for generating answers. Despite some issues in my initial code, the results demonstrated potential, and I shared my implementation with my group, emphasizing its efficiency and the necessity of fine-tuning for improved performance. This formed the foundation for our collaborative efforts, where I focused on embedding generation, cosine similarity for document retrieval, and integrating BART into our workflow. Additionally, I designed the initial page of our Streamlit application, showcasing the fine-tuning approach. Subsequently, I collaborated with my group to incorporate other methods and refine the system further.

```
chunk_size = 100
chunks = []

for speech in data['transcript']:
    words = speech.split()
    for i in range(0, len(words), chunk_size):
        chunks.append(" ".join(words[i:i + chunk_size]))

vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = vectorizer.fit_transform(chunks)


def get_most_relevant_chunk(question, top_n=2, similarity_threshold=0.5):
    question_vector = vectorizer.transform([question])
    similarities = cosine_similarity(question_vector, tfidf_matrix)

    top_n_indices = np.argsort(similarities[0])[-top_n:][::-1]

    highest_similarity = similarities[0][top_n_indices[0]]
    if highest_similarity < similarity_threshold:
        return None, False

    combined_context = " ".join([chunks[idx] for idx in top_n_indices])
    return combined_context, True
```

Fig.7: Cosine/Vectorization in main code

Main work: A SentenceTransformer model generates embeddings for speeches stored in a Chroma database. This allows retrieval of relevant excerpts based on user questions using similarity search. Answers are summarized using a BART-based pipeline, and sentiment analysis is performed using a DistilBERT model, with visual sentiment feedback via emojis.

```
summarizer = pipeline("summarization", model="facebook/bart-large-cnn", device=0 if t


def summarize_response(response):
    if len(response.split()) <= 20:
        return response

    summarized = summarizer(response, max_length=50, min_length=10, do_sample=False)
    return summarized[0]['summary_text']
```

Fig.8: Summarization using Facebook Bart

Although my primary contributions were in Embedding and Retrieval, Summarization, Sentiment Analysis, and Streamlit integration, I actively collaborated with my group members on other tasks to ensure seamless synchronization and alignment across all aspects of the project.

As per the calculation approximated 65% of code was generated from internet.

Conclusion:

This project successfully implemented a system for analyzing presidential speeches, providing users with tools for question answering, summarization, and sentiment analysis. Through the integration of embedding-based retrieval, summarization techniques, and sentiment evaluation, we achieved a streamlined and efficient workflow, accessible through a user-friendly Streamlit interface. Collaboration and effective task management ensured that all components were seamlessly aligned, contributing to the overall success of the project.
The application not only meets its initial objectives but also demonstrates the potential of NLP techniques in extracting insights from historical and political texts. By leveraging modern computational approaches, this project provides an innovative way to engage with and analyze complex datasets.

Future Improvements:

While the project achieved its goals, there are several avenues for future enhancements:

1. Enhanced Retrieval Accuracy: Incorporate more advanced retrieval models, such as fine-tuned transformer-based models like Dense Passage Retrieval (DPR), to improve the precision of responses.
2. Extended Sentiment Analysis: Expand sentiment analysis by incorporating more nuanced models that capture complex emotions and context-specific sentiment in political discourse.

3. Multilingual Support: Extend the system to support multilingual analysis, enabling insights from speeches in other languages to broaden the scope of the dataset.
4. Interactive Visualizations: Add dynamic visualizations for data exploration, such as sentiment trends over time, or heatmaps, to enhance user engagement.
5. Speech-to-Text Integration: Include a speech-to-text module to process audio recordings of speeches, enabling real-time analysis.
6. Deeper Contextual Analysis: Implement models for detecting rhetorical devices, identifying biases, and analyzing the evolution of themes in speeches over time.

References:

[1].https://www.reddit.com/r/learnmachinelearning/comments/12cp2cg/why_cosine_similarity_for_transformer_text/

[2]. https://spencerporter2.medium.com/understanding-cosine-similarity-and-word-embeddings-dbf19362a3c

[3]. https://christianbernecker.medium.com/nlp-summarization-use-bert-and-bart-to-summarize-your-favourite-newspaper-articles-fb9a81bed016

[4].https://www.reddit.com/r/LanguageTechnology/comments/iqgygo/what_is_currently_the_best_model_for/

[5]. https://medium.com/@lidores98/finetuning-huggingface-facebook-bart-model-2c758472e340

[6]. https://openai.com/index/better-language-models/