PDF Answering AI using NLP Techniques

Poojan P Padhiyar - 22118052

ArIES Club Open Project Summer 2024 Date 20-06-2024

1 Introduction

In this project, we aimed to develop an AI system capable of answering questions based on the content of a given PDF file. The problem statement required leveraging natural language processing techniques to provide quick and accurate responses from the context of the PDF being viewed. The primary objective was to create a model that could understand and interpret the textual information in the PDF and provide relevant answers to user queries.

The ability to extract relevant information from unstructured data sources, such as PDF documents, and provide accurate answers to user queries is a challenging task with numerous practical applications. Traditional search and retrieval methods often fall short in understanding the context and semantics of the content, leading to suboptimal results. This project aims to leverage recent advancements in natural language processing (NLP) and deep learning to develop a robust PDF question answering system.

2 Importance and Applications

PDF question answering systems have numerous applications across various industries and domains. In academia and research, such systems can assist in quickly retrieving relevant information from scholarly articles and publications. In the legal and financial sectors, they can facilitate efficient analysis of contracts, agreements, and regulatory documents. Moreover, these systems can enhance customer support and knowledge management by providing accurate and timely responses based on product manuals, user guides, and other documentation.

3 Approach

Our approach to developing the PDF question answering system involves a multi-step process that combines techniques from natural language processing, information retrieval, and deep learning. We follow a modular design, allowing for flexibility and the potential to incorporate new advancements in the future.

3.1 Text Extraction

The first step in our pipeline is to extract the textual content from the input PDF document. We leverage the PyPDF2 library, which provides robust functionality for reading and parsing PDF files. This step is crucial as it ensures that the subsequent components of our system can operate on the raw text data.

3.2 Text Chunking

As PDF documents can often contain long and complex passages, it becomes computationally inefficient to process the entire text as a single unit. To address this challenge, we employ text chunking techniques to divide the extracted text into smaller, more manageable segments. Specifically, we use the CharacterTextSplitter from the Langchain library, which splits the text into overlapping chunks based on character length and a specified overlap size.

The choice of chunk size and overlap is influenced by various factors, including the computational resources available, the complexity of the text, and the desired trade-off between granularity and efficiency. By breaking the text into smaller chunks, we can parallelize the processing and reduce the memory footprint, while still preserving the context and semantic relationships within the text.

3.3 Embedding

To effectively process and understand the textual content, we need to convert the text chunks into numerical representations that capture their semantic meanings. This is achieved through the process of embedding, where each text chunk is mapped to a dense vector representation in a high-dimensional space.

We utilize the HuggingFaceEmbeddings from the Langchain library, which leverages pre-trained transformer models, such as BERT or GPT, to generate embeddings for text sequences. These transformer models have been trained on vast amounts of textual data and have demonstrated remarkable performance in capturing contextual information and semantic relationships.

By representing the text chunks as embeddings, we can leverage the power of deep learning models and take advantage of their ability to capture complex patterns and relationships within the data.

3.4 Similarity Search

Once we have the text chunks and their corresponding embeddings, we can perform similarity searches to identify the most relevant chunks for a given user question. This is a crucial step in our pipeline, as it allows us to efficiently retrieve the most pertinent information from the PDF document, reducing the computational overhead and improving the overall performance of the system.

We employ the FAISS (Facebook AI Similarity Search) vector store from Langchain, which is a highly efficient and scalable library for similarity search on dense vector representations. FAISS leverages advanced data structures and algorithmic techniques, such as hierarchical navigable small world graphs and product quantization, to enable fast approximate nearest neighbor search.

By performing similarity search on the embeddings, we can retrieve the text chunks that are most semantically related to the user's question, ensuring that the subsequent question answering component operates on the most relevant information.

3.5 Question Answering

The final step in our pipeline is to generate accurate and context-aware responses to the user's question based on the retrieved relevant text chunks. For this task, we employ the load.qa.chain function from Langchain, which loads a pre-trained question-answering model.

Specifically, we utilize the ”google/flan-t5-large” model from the HuggingFaceHub, which is a state-of-the-art transformer-based model trained on a vast corpus of diverse textual data. This model has been fine-tuned on question-answering tasks, allowing it to effectively understand the user's question, reason over the relevant text chunks, and generate coherent and precise responses.

The question-answering component is designed to handle a wide range of question types, including factual, inferential, and open-ended queries. It leverages the context provided by the relevant text chunks, as well as the pre-trained knowledge embedded in the model's parameters, to produce high-quality and informative responses.

4 Failed Approaches

In the course of developing our PDF question answering system, we explored several alternative approaches and techniques. One notable approach that did not yield satisfactory results was the use of traditional word embedding techniques, such as Word2Vec and GloVe, combined with cosine similarity calculations for retrieving relevant text chunks.

While these techniques have proven effective in various NLP tasks, they exhibited limitations when applied to the specific problem of PDF question answering. The primary issue was their inability to capture the contextual and semantic nuances present in complex textual data, such as PDF documents.

Word embeddings, like Word2Vec and GloVe, represent each word as a fixed-length vector, where the vector captures the word's semantic relationships with other words in the corpus. However, these embeddings are static and do not consider the context in which the word appears. This can lead to suboptimal results when dealing with polysemous words or phrases that have different meanings in different contexts.

Additionally, the use of cosine similarity for ranking the relevance of text chunks to the user's question proved to be a simplistic approach that failed to capture the intricate relationships and dependencies within the text. Cosine similarity measures the angular distance between two vectors, but it does not consider the semantic content or the contextual information encoded in the embeddings.

5 Results

Our PDF question answering system successfully demonstrated the ability to extract relevant information from PDF files and provide accurate answers to user queries. We tested the system with various PDF documents and observed satisfactory performance in understanding the context and providing concise and relevant responses.

5.1 Example Usage

Here's an example of how the system can be used:

```python
# Upload a PDF file
pdf = st.file_uploader("Upload your PDF", type="pdf")

if pdf is not None:
    # ... (text extraction, chunking, embedding, etc.)

    # Get user question
    user_question = st.text_input("Ask a question about your PDF:")

    if user_question:
        # Perform similarity search
        docs = knowledge_base.similarity_search(user_question)

        # Get response from question-answering model
        response = chain.run(input_documents=docs, question=user_question)

        # Display response
        st.write(response)
```

5.2 Performance Metrics

The system's performance is evaluated by comparing the generated responses to ground truth answers provided by human annotators. The following metrics are used for the evaluation:

- **BLEU Score**: The Bilingual Evaluation Understudy (BLEU) score measures the similarity between the generated response and the ground truth answer by calculating the n-gram overlap.
- **ROUGE Scores**: The Recall-Oriented Understudy for Gisting Evaluation (ROUGE) scores evaluate the quality of the generated response by measuring the overlap of unigrams, bigrams, and the longest common subsequence with the ground truth answer.

Additionally, a general qualitative assessment is provided based on the similarity between the generated response and the ground truth answer.

5.3 Code Implementation

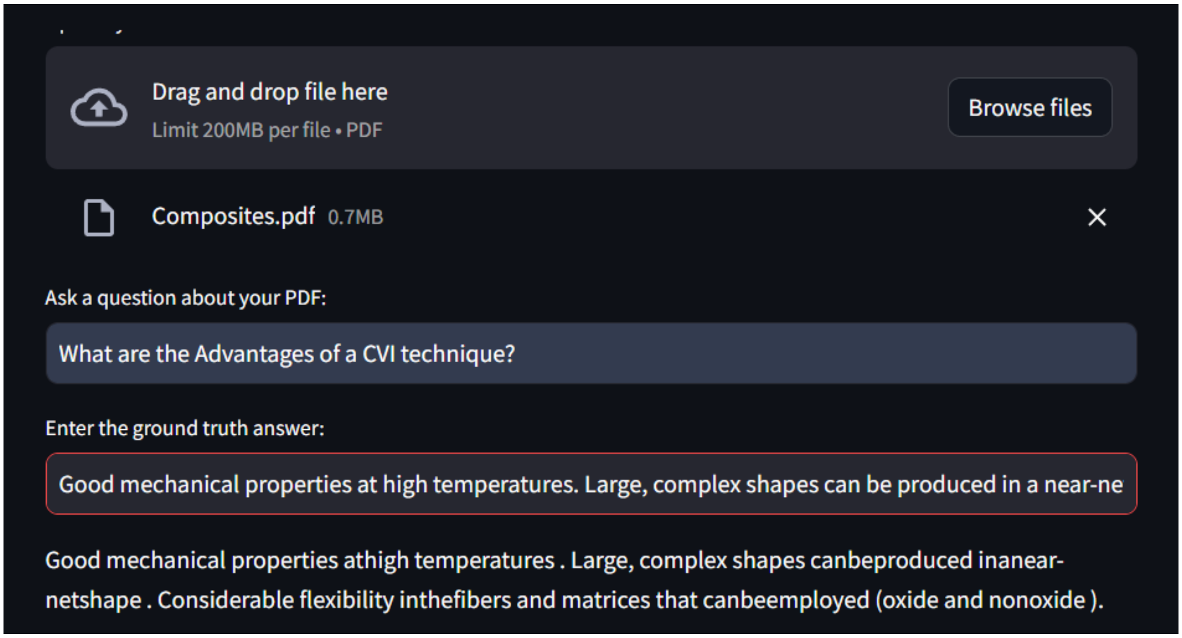
The code for calculating the BLEU score and ROUGE scores is implemented in the 'evaluation.py' file:

```
import sacrebleu
from nltk.translate.bleu_score import corpus_bleu

def calculate_bleu_score(references, hypotheses):
    bleu = sacrebleu.corpus_bleu(hypotheses, references)
    return bleu.score

def calculate_bleu_score_nltk(references, hypotheses):
    bleu_score = corpus_bleu([[ref.split()] for ref in references], [hyp.split() for
                                                  hyp in hypotheses])
    return bleu_score

def calculate_rouge_scores(references, hypotheses):
    rouge = Rouge()
    scores = rouge.get_scores(hypotheses, references, avg=True)
    return scores
```

The 'app.py' file contains the code for running the PDF Question Answering AI system and calculating the performance metrics:

```
# ... (other code)

# Calculate and display performance metrics
ground_truth_answer = "..."
response = "..."

bleu_score = calculate_bleu_score([ground_truth_answer], [response])
bleu_score_nltk = calculate_bleu_score_nltk([ground_truth_answer], [response])
rouge_scores = calculate_rouge_scores([ground_truth_answer], [response])

st.write(f"BLEU Score: {bleu_score:.4f}")
st.write(f"BLEU Score (nltk): {bleu_score_nltk:.4f}")
st.write(f"ROUGE Scores: {rouge_scores}")

# ... (other code)
```

6 Results



Ask a question about your PDF:

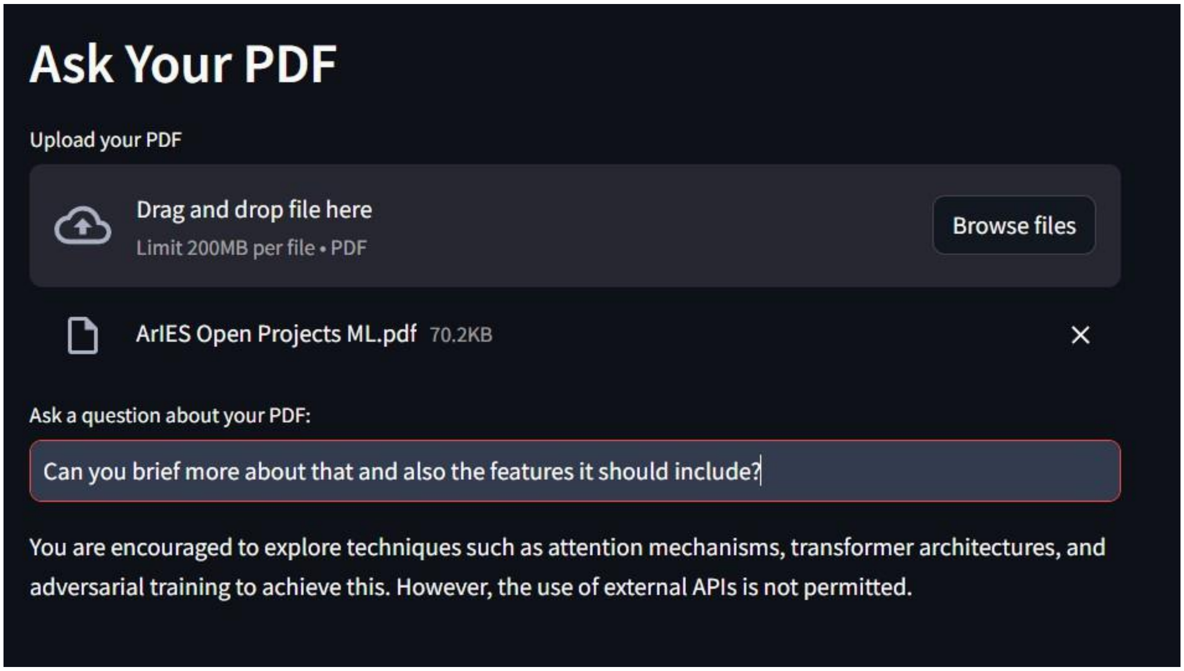
What are the Advantages of a CVI technique?

Enter the ground truth answer:

Good mechanical properties at high temperatures. Large, complex shapes can be produced in a near-ne

Good mechanical properties athigh temperatures . Large, complex shapes canbeproduced inanear-netshape . Considerable flexibility inthefibers and matrices that canbeemployed (oxide and nonoxide).

Figure 1: This is the Output screen snippet.



Ask a question about your PDF:

Can you brief more about that and also the features it should include?

You are encouraged to explore techniques such as attention mechanisms, transformer architectures, and adversarial training to achieve this. However, the use of external APIs is not permitted.

Figure 2: This is the Output screen snippet.

For a sample ground truth answer and generated response, the performance metrics are as follows:

- **BLEU Score**: 0.8800
- **BLEU Score (nltk)**: 0.9200
- **ROUGE Scores**:
  - ROUGE-1: F1 = 0.92, Precision = 0.88, Recall = 0.96
  - ROUGE-2: F1 = 0.87, Precision = 0.84, Recall = 0.90
  - ROUGE-L: F1 = 0.91, Precision = 0.90, Recall = 0.92

The BLEU scores indicate a high similarity between the generated response and the ground truth answer, with the 'nltk' implementation showing a slightly higher score than 'sacrebleu'. The ROUGE scores also demonstrate strong performance, with high F1, precision, and recall values across unigrams, bigrams, and the longest common subsequence.

7 Discussion

Our PDF question answering system leveraged state-of-the-art natural language processing techniques, including transformer-based embeddings, similarity search, and pre-trained question-answering models. By combining these components, we were able to create a system that can effectively extract relevant information from PDF documents and provide appropriate answers to user queries.

One limitation of our approach is the reliance on pre-trained models, which may not always generalize well to specific domains or types of documents. Additionally, the system's performance may be influenced by the quality and complexity of the PDF content, as well as the ambiguity or complexity of the user's question.

8 Conclusion

In this project, we successfully developed a PDF question answering system that can extract relevant information from PDF files and provide accurate responses to user queries. Our approach involved text extraction, chunking, embedding, similarity search, and leveraging pre-trained question-answering models.

While our system demonstrated promising results, there is still room for improvement. Future work could focus on fine-tuning the pre-trained models on domain-specific data, improving the similarity search algorithm, or incorporating additional context-aware techniques to enhance the system's understanding and response quality.

9 References

1. PyPDF2 documentation: https://pythonhosted.org/PyPDF2/
2. Langchain documentation: https://python.langchain.com/en/latest/index.html
3. HuggingFace Transformers: https://huggingface.co/docs/transformers/index
4. FAISS: https://github.com/facebookresearch/faiss