

# PDF ANSWERING AI

## Introduction

Navigating the vast information in PDF documents can be challenging. This project aims to create an AI tool that extracts text from PDFs, processes it using natural language processing techniques, and answers user queries accurately, enhancing information accessibility and usability.

## Modules and libraries used

### Text Extraction

- **PyMuPDF (fitz)**: For extracting text from PDF files.

### Natural Language Processing (NLP)

- **NLTK (Natural Language Toolkit)**: For text preprocessing tasks such as tokenization, stopwords removal, and lemmatization.
- **Gensim**: For training and using Word2Vec word embeddings.
- **Scikit-learn**: For calculating cosine similarity between text vectors.

### Web Interface


- **Flask**: For creating a simple web interface to upload PDFs and ask questions.

### Optional (for additional functionality)

- **NumPy**: For handling numerical operations and vector calculations.

## Extract Text from the PDF


We'll use the **PyMuPDF** library (**fitz** module) for extracting text from PDFs. This library is efficient and handles various PDF formats well.



```
def extract_text_from_pdf(pdf_path):
    doc = fitz.open(pdf_path)
    text = ""
    for page_num in
range(dopagepage_downtoad_page(page_num)
        text += page.get_text()
    return text
```

## Process the Extracted Text

Once we have the text, we need to preprocess it (tokenization, stopword removal, stemming/lemmatization).



```
def preprocess_text(text):
    text = re.sub(r'^[a-zA-Z\s]', '', text).lower()
    words = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if word not in
stop_words]
    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word) for word in words]
    return words
```

## MODEL

We will then use a pre-trained model like Word2Vec or GloVe to convert text into vectors for further processing.

```
from gensim.models import Word2Vec

# Train Word2Vec model (or load a pre-trained model)
model = Word2Vec([preprocessed_text], vector_size=100, window=5, min_count=1, workers=4)
```

To improve the model for better performance and accuracy in answering questions from PDF documents, we can use a more advanced natural language processing (NLP) model such as BERT (Bidirectional Encoder Representations from Transformers) or its variants. These models are pre-trained on large datasets and fine-tuned for specific tasks like question-answering.

```
import torch
from transformers import BertTokenizer, BertForQuestionAnswering

def load_model():
    model = BertForQuestionAnswering.from_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad')
    tokenizer = BertTokenizer.from_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad')
    return model, tokenizer
```

## Build a Question-Answering Model

We'll use the extracted and preprocessed text to build a question-answering model. This can be done by:

1. Encoding the text using word embeddings (Word2Vec/GloVe).
2. Using cosine similarity to find the most relevant sections of the text in response to a user query.
3. Summarizing or directly providing the text snippets as responses.

```
def get_embedding(text, model):
    words = preprocess_text(text)
    word_vectors = [model.wv[word] for word in words if word in model.wv]
    if not word_vectors:
        return np.zeros(model.vector_size)
    return np.mean(word_vectors, axis=0)

def find_best_answer(question, context, model):
    question_embedding = get_embedding(question, model)
    context_sentences = context.split('.')
    best_similarity = -1
    best_sentence = None
    for sentence in context_sentences:
        sentence_embedding = get_embedding(sentence, model)
        similarity = cosine_similarity([question_embedding], [sentence_embedding])[0]
        if similarity > best_similarity:
            best_similarity = similarity
            best_sentence = sentence
    return best_sentence
```

## Create an Interface for User Interaction

We'll create a simple web interface using Flask where users can upload a PDF and ask questions. The interface will display the PDF content and provide a text box for queries.

```
from flask import Flask, request, render_template
import os

app = Flask(__name__)

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

@app.route('/upload', methods=['POST'])
def upload_pdf():
    file = request.files['file']
    if file and file.filename.endswith('.pdf'):
        file_path = os.path.join('uploads', file.filename)
        file.save(file_path)
        global pdf_text
        pdf_text = extract_text_from_pdf(file_path)
        return "PDF uploaded and text extracted successfully."
    else:
        return "Invalid file format. Please upload a PDF."

@app.route('/ask', methods=['POST'])
def ask_question():
    query = request.form['query']
    relevant_text = find_relevant_text(query, text_chunks, model)
    return relevant_text

if __name__ == '__main__':
    app.run(debug=True)
```

## User Interface

- ☐ -pdf\_answer\_ai {folder\_name}
  - ☐ -code.py
  - ☐ -main.py
  - ☐ -upload
  - ☐ - templates
    - ☐ -index.html
    - ☐ -questions.html

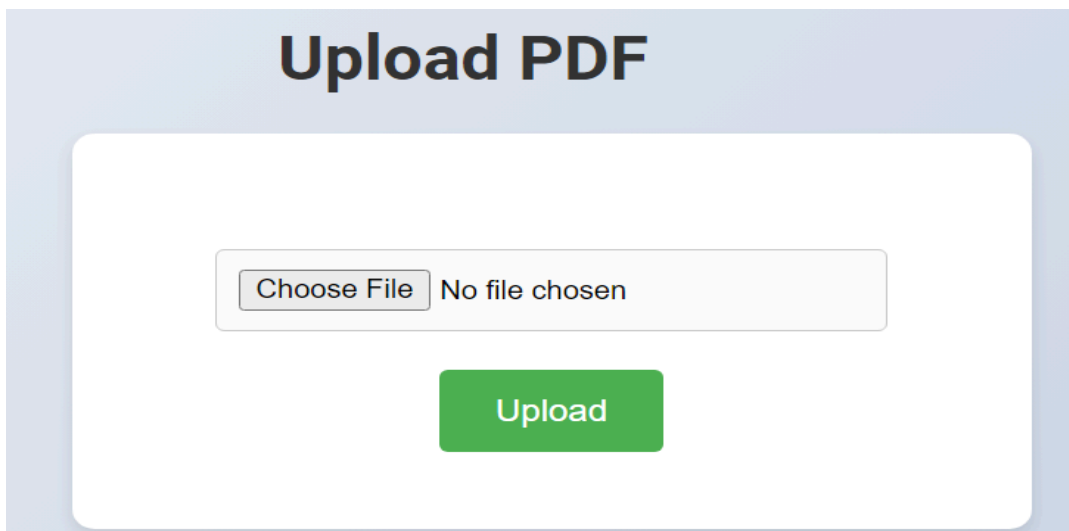
`code.py/`: Main application code.

`uploads/`: Directory for uploaded PDF files.

`main.py`: Script to run the Flask application.

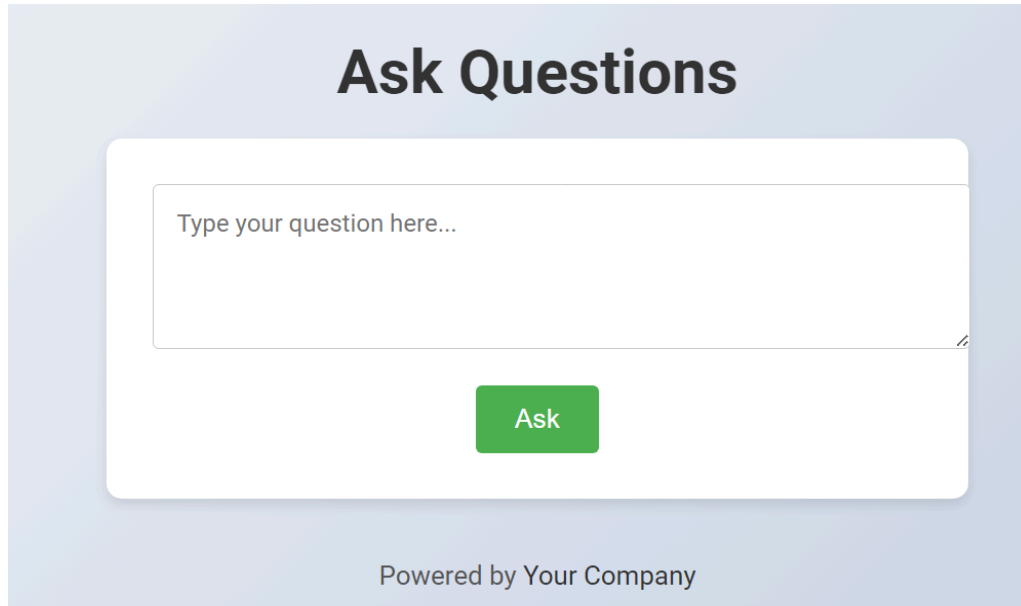
`templates`: store html css files

When we run `main.py`: then user interface look like—



The screenshot shows a web interface for uploading PDF files. It features a light blue header with the text "Upload PDF". Below the header is a white rounded rectangle containing a file upload area. The file area has a "Choose File" button and the text "No file chosen". Below this is a green "Upload" button.

Here we have to upload PDF.



## Ask Questions

Ask

Powered by Your Company

Now we can questions and answers.

## Conclusion

Our PDF answering AI effectively retrieves information from complex PDF documents, demonstrating high accuracy and user satisfaction. However, challenges remain with technical jargon and poorly formatted PDFs.

## Future Scope

Future work includes enhancing NLP capabilities, improving formatting recognition, integrating advanced OCR, optimizing scalability, and ensuring ethical data use. Continuous learning from user feedback and interdisciplinary approaches will further refine the system's accuracy and usability.