Streamlit PDF Question-Answering Chatbot Documentation

Debajyoti Maity

November 4, 2024

1 Introduction

This document provides a detailed explanation of the Streamlit application for PDF-based question-answering. The application enables users to upload a PDF document, select specific pages to include in the context, and ask questions based on the content of the selected pages. The responses are generated using a placeholder function that can be replaced with a real AI model.

2 App Initialization

The application starts by defining the `main()` function and setting the title of the app.

```
def main():
    st.title(" PDF Question-Answering Chatbot")
```

Listing 1: App Initialization

Explanation:

- `main()`: This function initializes the Streamlit app.
- `st.title`: Sets the title of the app that appears at the top of the browser page.

3 Sidebar for File Upload and Page Selection

This section creates a sidebar in the app for PDF file upload and page selection.

```
st.sidebar.header("PDF Upload and Page Selection")
uploaded_file = st.sidebar.file_uploader("Upload your PDF
    document here", type=['pdf'])
```

Listing 2: Sidebar for File Upload

Explanation:

- `st.sidebar.header`: Adds a header in the sidebar.
- `st.sidebar.file_uploader`: Creates a file uploader widget in the sidebar, allowing users to upload only PDF files.

4 File Processing

When a file is uploaded, a temporary file is created for processing.

```
if uploaded_file:
    with tempfile.NamedTemporaryFile(delete=False, suffix='.
        pdf') as tmp:
        tmp.write(uploaded_file.getbuffer())
        temp_pdf_path = tmp.name
```

Listing 3: File Processing

Explanation:

- `if uploaded_file`: Checks if a file has been uploaded.
- `tempfile.NamedTemporaryFile`: Creates a temporary file to store the uploaded PDF, allowing for safe and easy file manipulation.
- `tmp.write(uploaded_file.getbuffer())`: Writes the uploaded file's content to the temporary file.

5 PDF Reading and Page Selection

Reads the PDF file and allows the user to select pages to be included in the context.

```
pdfReader = PyPDF2.PdfReader(temp_pdf_path)
page_numbers = list(range(1, len(pdfReader.pages) + 1))
selected_pages = st.sidebar.multiselect("Select pages to
    include in context:", page_numbers, default=page_numbers)
```

Listing 4: PDF Reading and Page Selection

Explanation:

- `PyPDF2.PdfReader`: Reads the PDF file.
- `list(range(...))`: Generates a list of page numbers based on the number of pages in the PDF.
- `st.sidebar.multiselect`: Displays a multi-select dropdown for choosing pages to include in the context.

6 Text Extraction

Extracts text from the selected pages.

```
context_text, input_length = pdf_file_preprocessing(
    temp_pdf_path, selected_pages)
```

Listing 5: Text Extraction

Explanation:

- `pdf_file_preprocessing(...)`: Extracts text from the selected pages and returns the text and its length.

7 User Interaction for Question Input

Allows the user to input a question based on the extracted context.

```
if context_text:
    st.write("Context extracted. You can now ask a question
        based on the PDF content.")
    question = st.text_input("Enter your question here:")
    if st.button("Get Answer"):
        if question:
            with st.spinner("Generating answer..."):
                answer = generate_gemini_answer(context_text
                    , question)
            st.subheader("Answer:")
            st.write(answer)
        else:
            st.warning("Please enter a question.")
```

Listing 6: User Interaction for Question Input

Explanation:

- `st.text_input`: Provides an input box for the user to enter their question.
- `st.button`: When clicked, generates an answer based on the question and context.
- `st.spinner`: Displays a loading spinner while the answer is being generated.
- `st.subheader`: Displays the answer as a subheader.

8 Error Handling for No Extracted Text

Displays a warning message if no text is extracted from the PDF.

```
else:
    st.warning("No text could be extracted from the selected
        pages. Please check the PDF file or page selection."
       )
```

Listing 7: Error Handling

Explanation:

- `st.warning`: Shows a warning if no text is extracted.

9 Clean Up Temporary File

Deletes the temporary file created earlier to free up resources.

```
os.unlink(temp_pdf_path)
```

Listing 8: Temporary File Deletion

Explanation:

- `os.unlink`: Deletes the temporary file to clean up and free resources.

10 Run the App

The script checks if it is being run directly and then executes the `main()` function.

```
if __name__ == "__main__":
    main()
```

Listing 9: Run the App

Explanation:

- `if __name__ == "__main__"`: Ensures the script is run directly and not imported as a module.
- `main()`: Calls the main function to start the app.

11 Conclusion

This document provides a detailed explanation of each component of the Streamlit app code. You can modify and extend the app as needed to suit your use case. To run the app, use the following command:

```
streamlit run app.py
```