```
!pip install transformers
!pip install langchain
```

```
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0
Requirement already satisfied: yarl<2.0,>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>
Collecting jsonpatch<2.0,>=1.33 (from langchain-core<0.4.0,>=0.3.8->langchain)
  Downloading jsonpatch-1.33-py2.py3-none-any.whl.metadata (3.0 kB)
Requirement already satisfied: packaging<25,>=23.2 in /usr/local/lib/python3.10/dist-packages (from langchain-cor
Requirement already satisfied: typing-extensions>=4.7 in /usr/local/lib/python3.10/dist-packages (from langchain-
Collecting httpx<1,>=0.23.0 (from langsmith<0.2.0,>=0.1.17->langchain)
  Downloading httpx-0.27.2-py3-none-any.whl.metadata (7.1 kB)
Collecting orjson<4.0.0,>=3.9.14 (from langsmith<0.2.0,>=0.1.17->langchain)
  Downloading orjson-3.10.7-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (50 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 50.4/50.4 kB 3.4 MB/s eta 0:00:00
Collecting requests-toolbelt<2.0.0,>=1.0.0 (from langsmith<0.2.0,>=0.1.17->langchain)
  Downloading requests_toolbelt-1.0.0-py2.py3-none-any.whl.metadata (14 kB)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.10/dist-packages (from pydantic<3
Requirement already satisfied: pydantic-core==2.23.4 in /usr/local/lib/python3.10/dist-packages (from pydantic<3.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2->lang
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.10/dist-packages (from SQLAlchemy<3,>=1
Requirement already satisfied: anyio in /usr/local/lib/python3.10/dist-packages (from httpx<1,>=0.23.0->langsmith
Collecting httpcore==1.* (from httpx<1,>=0.23.0->langsmith<0.2.0,>=0.1.17->langchain)
  Downloading httpcore-1.0.6-py3-none-any.whl.metadata (21 kB)
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-packages (from httpx<1,>=0.23.0->langsmi
Collecting h11<0.15,>=0.13 (from httpcore==1.*->httpx<1,>=0.23.0->langsmith<0.2.0,>=0.1.17->langchain)
  Downloading h11-0.14.0-py3-none-any.whl.metadata (8.2 kB)
Collecting jsonpointer>=1.9 (from jsonpatch<2.0,>=1.33->langchain-core<0.4.0,>=0.3.8->langchain)
  Downloading jsonpointer-3.0.0-py2.py3-none-any.whl.metadata (2.3 kB)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio->httpx<1,>=0
Downloading langchain-0.3.2-py3-none-any.whl (1.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 46.6 MB/s eta 0:00:00
Downloading langchain_core-0.3.8-py3-none-any.whl (400 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 400.9/400.9 kB 30.8 MB/s eta 0:00:00
Downloading langchain_text_splitters-0.3.0-py3-none-any.whl (25 kB)
Downloading langsmith-0.1.131-py3-none-any.whl (294 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 294.6/294.6 kB 25.0 MB/s eta 0:00:00
Downloading tenacity-8.5.0-py3-none-any.whl (28 kB)
Downloading httpx-0.27.2-py3-none-any.whl (76 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.4/76.4 kB 5.9 MB/s eta 0:00:00
Downloading httpcore-1.0.6-py3-none-any.whl (78 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 78.0/78.0 kB 6.9 MB/s eta 0:00:00
Downloading jsonpatch-1.33-py2.py3-none-any.whl (12 kB)
Downloading orjson-3.10.7-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (141 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 141.9/141.9 kB 12.6 MB/s eta 0:00:00
Downloading requests_toolbelt-1.0.0-py2.py3-none-any.whl (54 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.5/54.5 kB 5.1 MB/s eta 0:00:00
Downloading jsonpointer-3.0.0-py2.py3-none-any.whl (7.6 kB)
Downloading h11-0.14.0-py3-none-any.whl (58 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.3/58.3 kB 5.3 MB/s eta 0:00:00
Installing collected packages: tenacity, orjson, jsonpointer, h11, requests-toolbelt, jsonpatch, httpcore, httpx,
  Attempting uninstall: tenacity
    Found existing installation: tenacity 9.0.0
    Uninstalling tenacity-9.0.0:
      Successfully uninstalled tenacity-9.0.0
Successfully installed h11-0.14.0 httpcore-1.0.6 httpx-0.27.2 jsonpatch-1.33 jsonpointer-3.0.0 langchain-0.3.2 la
```

```
pip install PyPDF2
```

```
Collecting PyPDF2
  Downloading pypdf2-3.0.1-py3-none-any.whl.metadata (6.8 kB)
Downloading pypdf2-3.0.1-py3-none-any.whl (232 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 232.6/232.6 kB 10.8 MB/s eta 0:00:00
Installing collected packages: PyPDF2
Successfully installed PyPDF2-3.0.1
```

```
import PyPDF2
```

```
def extract_text_from_pdf(pdf_file):
    with open(pdf_file, 'rb') as file:
        reader = PyPDF2.PdfReader(file)
        text = ""
        for page in reader.pages:
            text += page.extract_text()
        return text
```

```
from transformers import pipeline
```

```
model = "distilbert-base-cased-distilled-squad"
qa_pipeline = pipeline('question-answering', model=model)
```

```
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_to
  warnings.warn(
```

```
def ask_question_from_pdf(question, context):
    response = qa_pipeline(question=question, context=context)
    return response['answer']
```

```
def main(pdf_file, question):
    # Extract text from PDF
    pdf_text = extract_text_from_pdf(pdf_file)

    # Ask a question from the extracted text
    answer = ask_question_from_pdf(question, pdf_text)

    return answer
```

```
pdf_file_path = "/content/Lecture 2- Intro to Hugging Face.pdf"
question = "What is the main topic of the document?"

answer = main(pdf_file_path, question)
print(f"Answer: {answer}")
```

```
Answer: State of transfer learning
```

Start coding or generate with AI.

Now this is working fine, Let us see an example by using gradio in more interactive Manner

```
pip install gradio
```

```
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas<3.0,>=1.0->gr
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas<3.0,>=1.0->
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.10/dist-packages (from pydantic>=
Requirement already satisfied: pydantic-core==2.23.4 in /usr/local/lib/python3.10/dist-packages (from pydantic>=2
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.10/dist-packages (from typer<1.0,>=0.12->gr
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from typer<1.0,>=0.
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.10/dist-packages (from typer<1.0,>=0.12->g
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->ma
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich>=10.11
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich>=10.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0-
Downloading gradio-4.44.1-py3-none-any.whl (18.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.1/18.1 MB 30.9 MB/s eta 0:00:00
Downloading gradio_client-1.3.0-py3-none-any.whl (318 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 318.7/318.7 kB 10.6 MB/s eta 0:00:00
Downloading tomlkit-0.12.0-py3-none-any.whl (37 kB)
Downloading aiofiles-23.2.1-py3-none-any.whl (15 kB)
Downloading fastapi-0.115.0-py3-none-any.whl (94 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 94.6/94.6 kB 6.8 MB/s eta 0:00:00
Downloading python_multipart-0.0.12-py3-none-any.whl (23 kB)
Downloading ruff-0.6.8-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (10.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.9/10.9 MB 62.4 MB/s eta 0:00:00
Downloading semantic_version-2.10.0-py2.py3-none-any.whl (15 kB)
Downloading uvicorn-0.31.0-py3-none-any.whl (63 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 63.7/63.7 kB 5.1 MB/s eta 0:00:00
Downloading ffmpy-0.4.0-py3-none-any.whl (5.8 kB)
Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
Downloading starlette-0.38.6-py3-none-any.whl (71 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 71.5/71.5 kB 5.7 MB/s eta 0:00:00
Downloading websockets-12.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux201
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 130.2/130.2 kB 10.6 MB/s eta 0:00:00
Installing collected packages: pydub, websockets, uvicorn, tomlkit, semantic-version, ruff, python-multipart, ffm
Successfully installed aiofiles-23.2.1 fastapi-0.115.0 ffmpy-0.4.0 gradio-4.44.1 gradio-client-1.3.0 pydub-0.25.1
```

```
import gradio as gr
from transformers import pipeline
import PyPDF2
```

```
# Load the question-answering model
qa_pipeline = pipeline("question-answering", model="distilbert-base-cased-distilled-squad")
```

```
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_to
  warnings.warn(
```

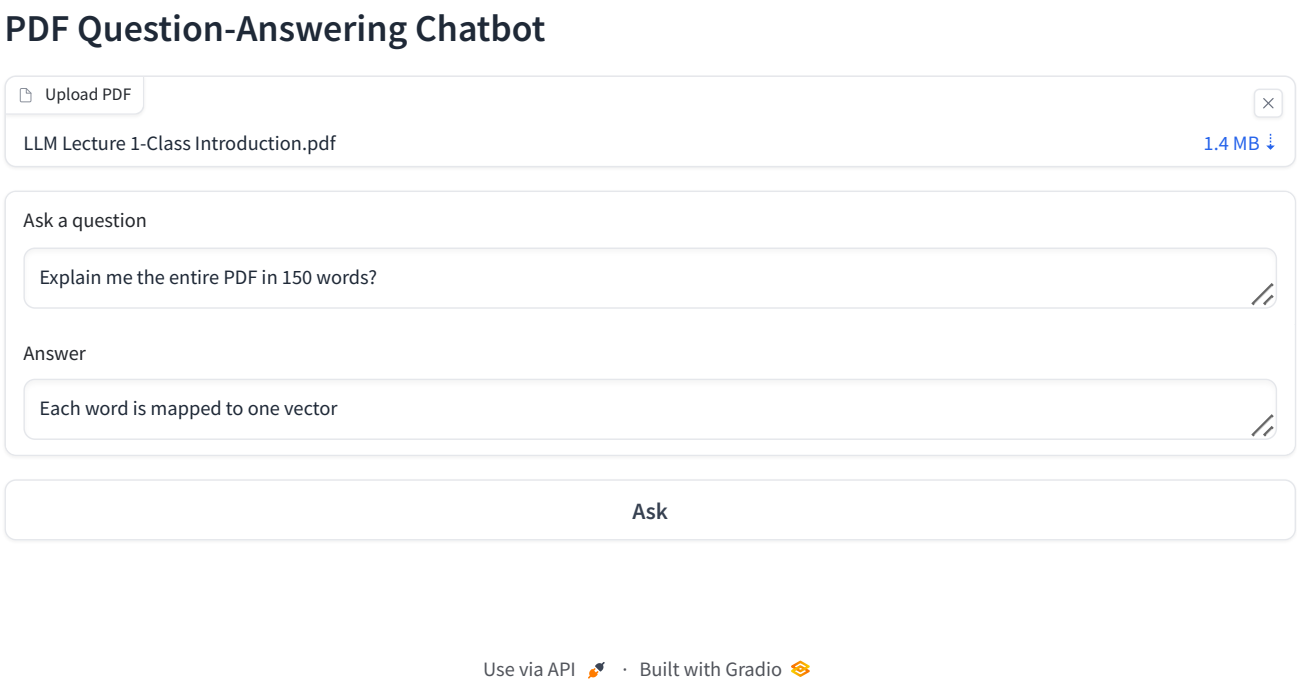
```
# Function to extract text from the PDF
def extract_text_from_pdf(pdf_file):
    reader = PyPDF2.PdfReader(pdf_file)
    text = ""
    for page in reader.pages:
        text += page.extract_text()
    return text
```

```
# Function to ask questions from the PDF context
def ask_question(question, pdf_file):
    # Extract text from the PDF
    pdf_text = extract_text_from_pdf(pdf_file)
    confidence_threshold = 0.3
    try:
        response = qa_pipeline(question=question, context=pdf_text)
        # Check if the answer's score is below the threshold
        if response['score'] < confidence_threshold:
            return "I cannot find the information"
        return response['answer']
    except:
        return "I cannot find the information"
```

```
with gr.Blocks() as demo:
    gr.Markdown("# PDF Question-Answering Chatbot")
```

```
    # File input for PDF upload
    pdf_input = gr.File(label="Upload PDF", type="filepath")

    # Text input for the user question
    question_input = gr.Textbox(label="Ask a question")

    # Output box for the answer
    answer_output = gr.Textbox(label="Answer")

    # Define how inputs are processed to produce output
    ask_button = gr.Button("Ask")
    ask_button.click(ask_question, inputs=[question_input, pdf_input], outputs=answer_output)

# Launch the Gradio interface
demo.launch(share=True)
```

```
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
Running on public URL: https://cec64a638fa60050de.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from Termin
```




```
# Function to ask questions from the PDF context
def ask_question(question, pdf_file):
    # Extract text from the PDF
    pdf_text = extract_text_from_pdf(pdf_file)
    confidence_threshold = 0.3
    try:
        response = qa_pipeline(question=question, context=pdf_text)
        # Check if the answer's score is below the threshold
        return response['answer']
    except:
        return "I cannot find the information"


# Create a Gradio interface
with gr.Blocks() as demo:
    # Title and description
    gr.Markdown(
        """
        # PDF Question-Answering Chatbot
        Upload a PDF document, ask a question, and the chatbot will answer based on the content of the document!
        """
    )
```

```
    
    # File input for PDF upload
    pdf_input = gr.File(label="Upload your PDF document here", type="filepath", file_types=[".pdf"])
    
    # Text input for the user question
    question_input = gr.Textbox(label="Type your question here", placeholder="Ask a question based on the uploaded PDF
    
    # Output box for the answer
    answer_output = gr.Textbox(label="Answer", placeholder="The answer will appear here", interactive=False, lines=3)
    
    # Button to trigger the question-answering function
    ask_button = gr.Button("Ask the Question")
    
    # Function to reset the PDF file upload
    reset_button = gr.Button("Reset PDF Upload")
    
    # Action on clicking the ask button
    ask_button.click(fn=ask_question, inputs=[question_input, pdf_input], outputs=answer_output)
    
    # Action on clicking the reset button
    reset_button.click(fn=lambda: None, inputs=None, outputs=pdf_input)
    
    # Footer
    gr.Markdown(
        """
        ### Instructions:
        - Upload your PDF document using the upload button.
        - Type your question in the text box below.
        - Click "Ask the Question" to get an answer based on the content of the PDF.
        - Use the "Reset PDF Upload" button to upload a new document.
        """
    )

# Launch the Gradio interface
demo.launch(share=True)
```

```
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
Running on public URL: https://a11d346ff26250fa48.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from Termin
```

What is the main topic of this PDF?

Answer

Large Language Model
Questions

Ask the Question

Reset PDF Upload

```
# Load a text generation model (e.g., GPT-2 or BART) for expanding or generating responses
text_generation_pipeline = pipeline("text-generation", model="gpt2")
```

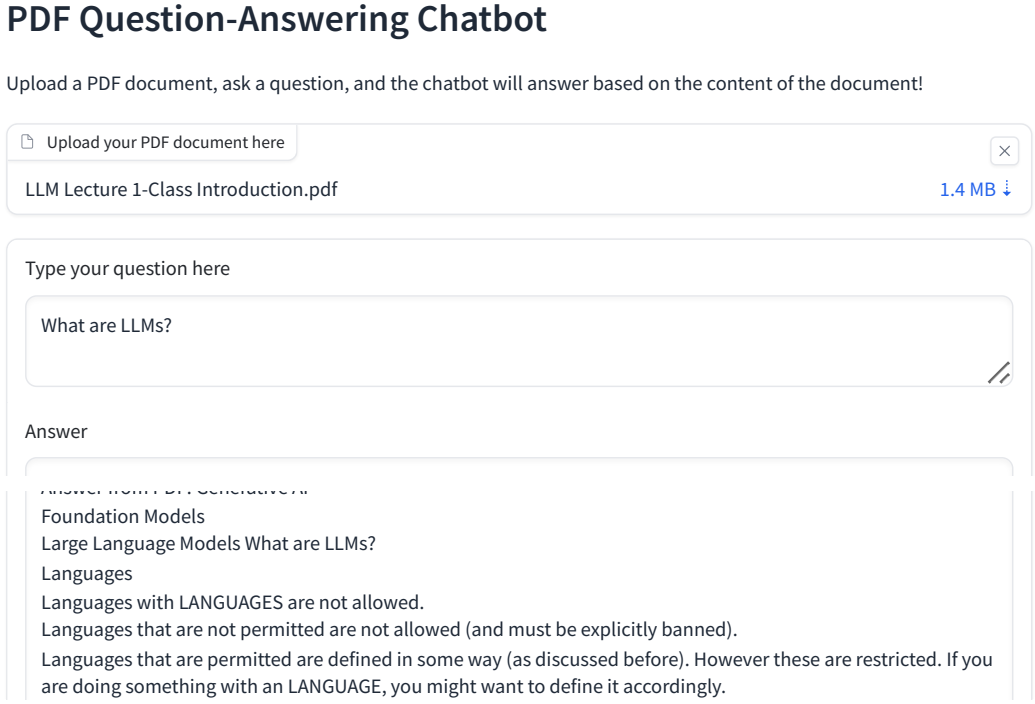
```
config.json: 100%                                   665/665 [00:00<00:00, 12.1kB/s]
model.safetensors: 100%                             548M/548M [00:13<00:00, 80.6MB/s]
generation_config.json: 100%                        124/124 [00:00<00:00, 1.77kB/s]
tokenizer_config.json: 100%                         26.0/26.0 [00:00<00:00, 702B/s]
vocab.json: 100%                                    1.04M/1.04M [00:00<00:00, 4.60MB/s]
merges.txt: 100%                                    456k/456k [00:00<00:00, 7.16MB/s]
tokenizer.json: 100%                                1.36M/1.36M [00:00<00:00, 7.66MB/s]
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_to
  warnings.warn(
```

```
# Function to handle asking questions
def ask_question(question, pdf_file_path):
    if not pdf_file_path:
        return "Please upload a PDF file."

    # Extract text from the PDF
    pdf_text = extract_text_from_pdf(pdf_file_path)

    # Set a confidence threshold for valid answers
    confidence_threshold = 0.3
    try:
        response = qa_pipeline(question=question, context=pdf_text)

        # If the model finds an answer with low confidence, fall back to the general model
        if response['score'] < confidence_threshold:
            # Use a fallback model to generate a broader response from general knowledge
            fallback_response = text_generation_pipeline(question, max_length=100)[0]['generated_text']
            return f"(PDF information not found, generating from model):\n\n{fallback_response}"

        # If the model finds an answer, expand on it using text generation
        expanded_answer = text_generation_pipeline(f"{response['answer']} {question}", max_length=100)[0]['generated_
        return f"Answer from PDF: {expanded_answer}"
    except Exception as e:
        # If there's an issue or no context found, fallback to a general answer from the model
        fallback_response = text_generation_pipeline(question, max_length=100)[0]['generated_text']
        return f"(Error or no relevant data in PDF, generating from model):\n\n{fallback_response}"
```

```
# Create a Gradio interface
with gr.Blocks() as demo:
    # Title and description
    gr.Markdown(
        """
        # PDF Question-Answering Chatbot
        Upload a PDF document, ask a question, and the chatbot will answer based on the content of the document!
        """
    )

    # File input for PDF upload
    pdf_input = gr.File(label="Upload your PDF document here", type="filepath", file_types=[".pdf"])

    # Text input for the user question
    question_input = gr.Textbox(label="Type your question here", placeholder="Ask a question based on the uploaded PD

    # Output box for the answer
    answer_output = gr.Textbox(label="Answer", placeholder="The answer will appear here", interactive=False, lines=3)

    # Button to trigger the question-answering function
    ask_button = gr.Button("Ask the Question")
```

```
    # Function to reset the PDF file upload
    reset_button = gr.Button("Reset PDF Upload")

    # Action on clicking the ask button
    ask_button.click(fn=ask_question, inputs=[question_input, pdf_input], outputs=answer_output)

    # Action on clicking the reset button
    reset_button.click(fn=lambda: None, inputs=None, outputs=pdf_input)

    # Footer
    gr.Markdown(
        """
        ### Instructions:
        - Upload your PDF document using the upload button.
        - Type your question in the text box below.
        - Click "Ask the Question" to get an answer based on the content of the PDF.
        - Use the "Reset PDF Upload" button to upload a new document.
        """
    )

# Launch the Gradio interface
demo.launch(share=True)
```

```
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
Running on public URL: https://7cedf9471cd119d198.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from Termin
```




```
##
```

Now I have changed the UserInterface to Chatbot

```
def ask_question(pdf_file_path, user_message, history):
    if not pdf_file_path:
        return "Please upload a PDF file.", history

    # Extract text from the PDF
    pdf_text = extract_text_from_pdf(pdf_file_path)

    # Set a confidence threshold for valid answers
    confidence_threshold = 0.3
```

```
    try:
        response = qa_pipeline(question=user_message, context=pdf_text)

        # If the model finds an answer with low confidence, fall back to the general model
        if response['score'] < confidence_threshold:
            fallback_response = text_generation_pipeline(user_message, max_length=100)[0]['generated_text']
            answer = f"**PDF information not found, generating from model:**\n\n{fallback_response}"
        else:
            # Expand the answer using text generation
            expanded_answer = text_generation_pipeline(f"{response['answer']} {user_message}", max_length=100)[0]['gen
            answer = f"**Answer from PDF:**\n\n{expanded_answer}"
    except Exception as e:
        fallback_response = text_generation_pipeline(user_message, max_length=100)[0]['generated_text']
        answer = f"**Error or no relevant data in PDF, generating from model:**\n\n{fallback_response}"

    # Update the chat history
    history.append((user_message, answer))
    return "", history  # Clear user message and return updated history


with gr.Blocks() as demo:
    # Title and description
    gr.Markdown(
        """
        # PDF Question-Answering Chatbot
        **Upload a PDF document and ask questions in a chat format!**
        The chatbot will provide answers based on the content of the document.
        If no information is found, it will generate a response from its knowledge base.
        """
    )

    # File input for PDF upload
    pdf_input = gr.File(label="Upload your PDF document here", type="filepath", file_types=[".pdf"])

    # Chatbot interface for user interaction
    chatbot = gr.Chatbot(label="Chat with the PDF Q&A Bot")

    # Text input for user message
    user_message_input = gr.Textbox(label="Type your question here", placeholder="Ask a question based on the uploade

    # Button to submit the user question
    submit_button = gr.Button("Send")

    # Initialize the chat history
    chat_history = []

    # Action on clicking the submit button
    submit_button.click(fn=ask_question, inputs=[pdf_input, user_message_input, chatbot], outputs=[user_message_input

    # Footer instructions
    gr.Markdown(
        """
        ### Instructions:
        - **Upload a PDF document** using the upload button.
        - **Type your question** in the text box and press "Send".
        - The chatbot will respond based on the content of the PDF or generate from its model if needed.
        """
    )
# Launch the Gradio interface
demo.launch()
```

```
Setting queue=True in a Colab notebook requires sharing enabled. Setting `share=True` (you can turn this off by s

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
Running on public URL: https://b2ae09955dd6d28c67.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from Termin
```

## PDF Question-Answering Chatbot

**Upload a PDF document and ask questions in a chat format!** The chatbot will provide answers based on the content of the document. If no information is found, it will generate a response from its knowledge base.

Upload your PDF document here

LLM Lecture 1-Class Introduction.pdf 1.4 MB

Chat with the PDF Q&A Bot

can you explain me what is LLMs?

**Answer from PDF:**

Generative AI
Foundation Models
Large Language Models can you explain me what is LLMs?
When do you build the foundations of your system? Do you generate the language models for your code base using a single language? Or do you do all of the compilation and test for each language?

Start coding or generate with AI.