Here's the **fully working Gradio app** for a PDF Q&A chatbot using:

- ✅ `langchain_ollama` for LLaMA 3 via Ollama
- ✅ `langchain_huggingface` for HuggingFace embeddings
- ✅ `FAISS` for vector storage
- ✅ `gradio` for the user interface
- ✅ Correct handling of file paths (no `.read()` bug)

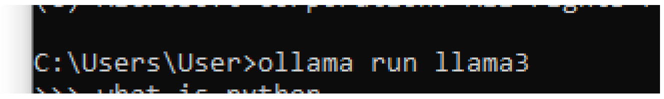
✅ Complete & Fixed Code (`chatbot.py`):-

```
import gradio as gr
from langchain_community.document_loaders import PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
from langchain_ollama import OllamaLLM
from langchain.chains import RetrievalQA

# Step 1: Load PDF and split into documents
def load_pdf(path):
    loader = PyPDFLoader(path)
    docs = loader.load_and_split()
    return docs

# Step 2: Create FAISS vector store from documents
def create_vectorstore(docs):
    splitter = RecursiveCharacterTextSplitter(chunk_size=500,
chunk_overlap=50)
    texts = splitter.split_documents(docs)
    embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-
MiniLM-L6-v2")
    vectordb = FAISS.from_documents(texts, embedding=embeddings)
    return vectordb

# Step 3: Load Ollama LLM
def load_llm():
    return OllamaLLM(model="llama3")

# Step 4: Build Retrieval QA Chain
def create_qa_chain(llm, vectordb):
    retriever = vectordb.as_retriever(search_kwargs={"k": 3})
    return RetrievalQA.from_chain_type(llm=llm, chain_type="stuff",
retriever=retriever)

# Handle PDF processing
```

```python
def process_pdf(file_path):
    try:
        docs = load_pdf(file_path)
        vectordb = create_vectorstore(docs)
        llm = load_llm()
        qa_chain = create_qa_chain(llm, vectordb)
        return f"✅ PDF processed. Ready to ask questions.", qa_chain
    except Exception as e:
        return f"❌ Error while processing PDF: {e}", None

# Handle Q&A interaction
def answer_question(question, qa_chain):
    if not qa_chain:
        return "⚠️ Please upload and process a PDF first."
    if not question.strip():
        return "⚠️ Please enter a question."

    try:
        result = qa_chain.invoke({"query": question})
        return result["result"] if isinstance(result, dict) else result
    except Exception as e:
        return f"❌ Error while answering: {e}"

# Build Gradio UI
with gr.Blocks() as demo:
    gr.Markdown("# 📄 Ask Questions About Your PDF")
    gr.Markdown("Built with **Ollama + LLaMA 3 + HuggingFace + FAISS**")

    qa_chain_state = gr.State()

    pdf_input = gr.File(label="📎 Upload a PDF", file_types=[".pdf"])
    upload_button = gr.Button("📤 Process PDF")
    status_output = gr.Textbox(label="Status", lines=2)

    user_question = gr.Textbox(label="💬 Ask a question about the PDF")
    ask_button = gr.Button("🤖 Get Answer")
    answer_output = gr.Textbox(label="📝 Answer", lines=4)

    upload_button.click(fn=process_pdf, inputs=pdf_input,
outputs=[status_output, qa_chain_state])
    ask_button.click(fn=answer_question, inputs=[user_question,
qa_chain_state], outputs=answer_output)

demo.launch()
```

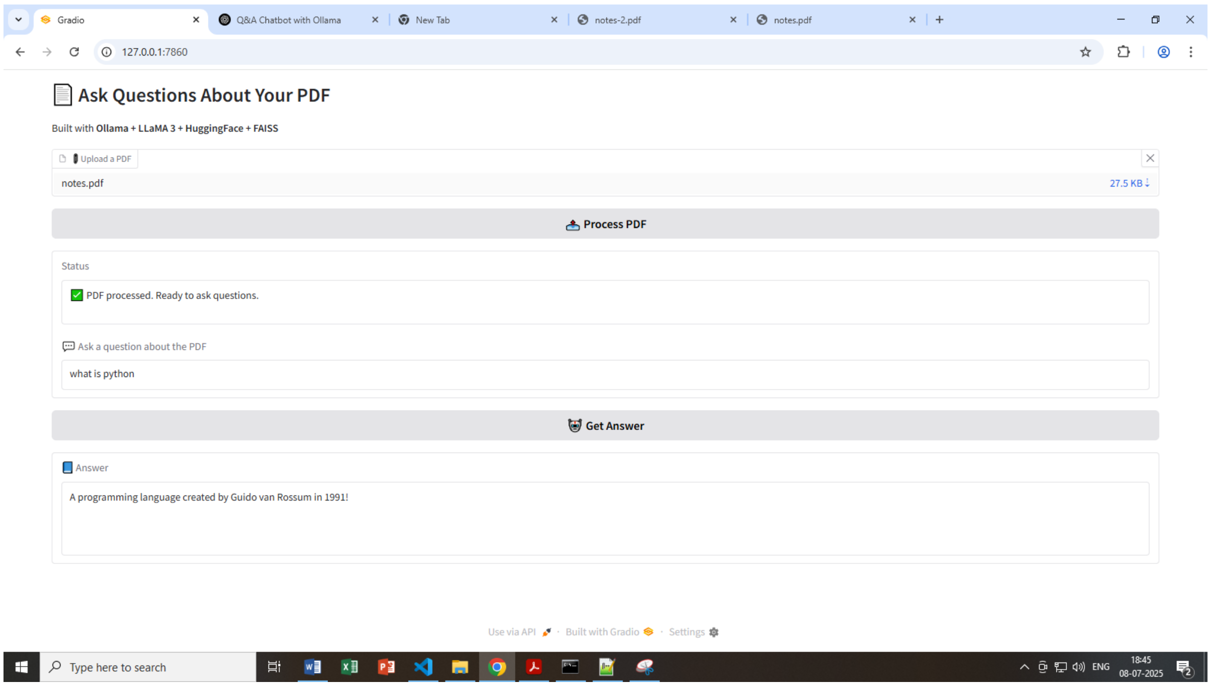
output:-

first run ollama then run chatbot.py file :-



```
C:\Users\User>ollama run llama3
```

http://127.0.0.1:7860/



📄 Ask Questions About Your PDF

Built with **Ollama + LLaMA 3 + HuggingFace + FAISS**

Upload a PDF

notes.pdf 27.5 KB

Process PDF

Status

✅ PDF processed. Ready to ask questions.

💬 Ask a question about the PDF

what is python

Get Answer

📘 Answer

A programming language created by Guido van Rossum in 1991!

Here's a **line-by-line explanation** of your PDF Q&A chatbot using **Gradio**, **LangChain**, **HuggingFace Embeddings**, **FAISS**, and **Ollama (LLaMA 3)**:

📦 Imports

```
import gradio as gr
```

→ Imports **Gradio** — the Python library to build interactive web UIs for AI apps.

```
from langchain_community.document_loaders import PyPDFLoader
```

➡️ Loads the `PyPDFLoader` to extract text from PDFs using LangChain's community-maintained tools.

```
from langchain.text_splitter import RecursiveCharacterTextSplitter
```

➡️ Used to **split long PDF text into smaller chunks** (based on characters) so the model can handle it.

```
from langchain_huggingface import HuggingFaceEmbeddings
```

➡️ Imports sentence embeddings from HuggingFace (used for semantic search).

```
from langchain_community.vectorstores import FAISS
```

➡️ FAISS is a fast vector database. It stores embeddings and enables efficient similarity search.

```
from langchain_ollama import OllamaLLM
```

➡️ Uses the **Ollama** local LLM interface to run `llama3` or any other local model.

```
from langchain.chains import RetrievalQA
```

➡️ Combines a retriever (FAISS) + LLM (Ollama) to build a **Question Answering chain**.

📄 PDF Processing Functions

1. Load the PDF and split it

```
def load_pdf(path):
    loader = PyPDFLoader(path)
    docs = loader.load_and_split()
    return docs
```

- `PyPDFLoader(path)` reads the PDF.
- `.load_and_split()` reads and divides it into sections.
- Returns the list of document chunks (LangChain `Document` objects).

---

2. Turn documents into vector embeddings

```
def create_vectorstore(docs):
    splitter = RecursiveCharacterTextSplitter(chunk_size=500,
chunk_overlap=50)
    texts = splitter.split_documents(docs)
```

- Splits documents into 500-character chunks with 50-character overlap.

- Useful for preserving context across splits.

```
    embeddings = HuggingFaceEmbeddings(model_name="sentence-
transformers/all-MiniLM-L6-v2")
```

- Loads a **small, fast** transformer model for text embeddings.

```
    vectordb = FAISS.from_documents(texts, embedding=embeddings)
    return vectordb
```

- Converts document chunks to vectors using embeddings and stores them in FAISS.

---

3. Load the Ollama-based LLM

```
def load_llm():
    return OllamaLLM(model="llama3")
```

- Loads the LLaMA 3 model from Ollama.
- Works offline if Ollama and the model are installed locally.

---

4. Build RetrievalQA Chain

```
def create_qa_chain(llm, vectordb):
    retriever = vectordb.as_retriever(search_kwargs={"k": 3})
```

- Retrieves top 3 most relevant text chunks based on query.

```
    return RetrievalQA.from_chain_type(llm=llm, chain_type="stuff",
retriever=retriever)
```

- Combines retriever + LLM into a pipeline:
  (1) Retrieve relevant chunks → (2) Ask LLM to generate answer from them)

🔁 Gradio Callback Functions

5. Handle PDF Upload and Processing

```
def process_pdf(file_path):
    try:
        docs = load_pdf(file_path)
        vectordb = create_vectorstore(docs)
        llm = load_llm()
        qa_chain = create_qa_chain(llm, vectordb)
```

- Loads the PDF → splits it → creates vector index → loads the model → builds QA chain.

```
        return f"✅ PDF processed. Ready to ask questions.", qa_chain
    except Exception as e:
        return f"❌ Error while processing PDF: {e}", None
```

- Returns status message + chain (stored in Gradio's `gr.State()`).

---

6. Handle Question-Answering

```
def answer_question(question, qa_chain):
    if not qa_chain:
        return "⚠️ Please upload and process a PDF first."
```

- Checks if QA chain is initialized.

```
    if not question.strip():
        return "⚠️ Please enter a question."
```

- Checks if question is empty.

```
    result = qa_chain.invoke({"query": question})
    return result["result"] if isinstance(result, dict) else result
```

- Uses `.invoke()` to get answer from chain and returns it.

🌐 Gradio UI

7. Build the interface

```
with gr.Blocks() as demo:
```

- Starts building a Gradio app.

```
    gr.Markdown("# 📄 Ask Questions About Your PDF")
    gr.Markdown("Built with **Ollama + LLaMA 3 + HuggingFace + FAISS**")
```

- Shows title and description.

```
    qa_chain_state = gr.State()
```

- Keeps the `qa_chain` saved between button clicks.

```
pdf_input = gr.File(label="📎 Upload a PDF", file_types=[".pdf"])
upload_button = gr.Button("📤 Process PDF")
status_output = gr.Textbox(label="Status", lines=2)
```

- File uploader for PDF.
- A button to trigger processing.
- A textbox to show processing result.

```
user_question = gr.Textbox(label="💬 Ask a question about the PDF")
ask_button = gr.Button("🤖 Get Answer")
answer_output = gr.Textbox(label="📝 Answer", lines=4)
```

- User enters question.
- Button to submit.
- Answer shown in textbox.

---

8. Link buttons to functions

```
    upload_button.click(fn=process_pdf, inputs=pdf_input,
outputs=[status_output, qa_chain_state])
```

- When "Process PDF" is clicked:
  - `process_pdf()` is called with the uploaded file path.
  - Returns status + QA chain (saved in state).

```
    ask_button.click(fn=answer_question, inputs=[user_question,
qa_chain_state], outputs=answer_output)
```

- When "Get Answer" is clicked:
  - Sends user question and saved QA chain.
  - Displays answer.

---

9. Launch the app

```
demo.launch()
```

- Starts the Gradio web server (by default: http://localhost:7860).

✅ Final Summary

This app does:

1. Load a PDF and split it into chunks.
2. Convert chunks into embeddings using HuggingFace.
3. Store in FAISS for similarity search.
4. Use LLaMA 3 (via Ollama) to generate answers from retrieved chunks.
5. Provide a UI to upload a PDF and ask questions about it.