Generative AI Course Study Chatbot

This notebook builds a study chatbot for the Generative AI course using:

- LangChain (LCEL)
- Retrieval-Augmented Generation (RAG)
- Course slides as a knowledge base
- Gradio GUI for interaction

The chatbot answers questions ONLY based on course materials.

Step 1: Install Dependencies

```
!pip -q install -U langchain langchain-community langchain-text-splitters
!pip -q install -U faiss-cpu sentence-transformers
!pip -q install -U transformers gradio
!pip -q install -U langchain-openai
```

Step 2: Imports & Setup

```
import os
import getpass
import gradio as gr

from transformers import pipeline
from langchain_community.llms import HuggingFacePipeline

from langchain_openai import ChatOpenAI

from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser
from langchain_core.runnables import RunnablePassthrough

from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
```

Step 3: Load Course Slides (Data)

```
!pip install pypdf
```

```
Requirement already satisfied: pypdf in /usr/local/lib/python3.12/dist-packages (6.5.0)
```

The functionality of this cell is to make sure the data is uploaded

```
import os
print(os.listdir("data"))
```

```
['HTU - CPD - GenAI - Module2-A.pdf', 'HTU - CPD - GenAI - Module6.pdf', 'HTU - CPD - GenAI - Module3.pdf', 'HTU - CPD - GenAI -
```

```
import os
import glob
from langchain_community.document_loaders import PyPDFLoader

DATA_DIR = "data"

pdf_files = glob.glob(os.path.join(DATA_DIR, "*.pdf"))
print(f"Found {len(pdf_files)} PDF files")

documents = []
```

```
for pdf in pdf_files:
    loader = PyPDFLoader(pdf)
    documents.extend(loader.load())

print(f"Loaded {len(documents)} pages from course slides")
```

```
Found 8 PDF files
Loaded 470 pages from course slides
```

Step 4: Text Splitting (Chunking)

```
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,
    chunk_overlap=200
)

splits = text_splitter.split_documents(documents)
print(f"Total chunks: {len(splits)}")
```

```
Total chunks: 471
```

Step 5: Embeddings + Vector Store (FAISS)

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

vectorstore = FAISS.from_documents(splits, embeddings)

retriever = vectorstore.as_retriever(
    search_type="similarity",
    search_kwargs={"k": 4}
)

print("Vector store created successfully")
```

```
/tmp/ipython-input-4022519102.py:1: LangChainDeprecationWarning: The class `HuggingFaceEmbeddings` was deprecated in LangChain 0
  embeddings = HuggingFaceEmbeddings(
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
modules.json: 100% 349/349 [00:00<00:00, 9.39kB/s]
config_sentence_transformers.json: 100% 116/116 [00:00<00:00, 6.47kB/s]
README.md: 10.5k/? [00:00<00:00, 932kB/s]
sentence_bert_config.json: 100% 53.0/53.0 [00:00<00:00, 2.22kB/s]
config.json: 100% 612/612 [00:00<00:00, 19.4kB/s]
model.safetensors: 100% 90.9M/90.9M [00:01<00:00, 83.3MB/s]
tokenizer_config.json: 100% 350/350 [00:00<00:00, 9.99kB/s]
vocab.txt: 232k/? [00:00<00:00, 4.32MB/s]
tokenizer.json: 466k/? [00:00<00:00, 8.69MB/s]
special_tokens_map.json: 100% 112/112 [00:00<00:00, 2.60kB/s]
config.json: 100% 190/190 [00:00<00:00, 5.60kB/s]
Vector store created successfully
```

Step 6:

6.1 OpenRouter API Setup

```
if not os.getenv("OPENROUTER_API_KEY"):
    os.environ["OPENROUTER_API_KEY"] = getpass.getpass("Enter your OpenRouter API key: ")

print("OpenRouter API key set:", bool(os.getenv("OPENROUTER_API_KEY")))
```

```
Enter your OpenRouter API key: ··········
OpenRouter API key set: True
```

6.2 OpenRouter LLM Setup

```
import os

llm_online = ChatOpenAI(
    model="mistralai/mistral-7b-instruct",
    base_url="https://openrouter.ai/api/v1",
    api_key=os.getenv("OPENROUTER_API_KEY"),
    temperature=0.5,
    max_tokens=100
)

print("Online LLM (OpenRouter) ready")
```

```
Online LLM (OpenRouter) ready
```

6.3 LLM Selector (Hybrid Switch)

```
USE_ONLINE = True

llm_used = llm_online if USE_ONLINE else llm_offline

print("LLM in use:", "OpenRouter" if USE_ONLINE else "Offline HF")
```

```
LLM in use: OpenRouter
```

Step 7: Prompt Template

```
rag_prompt = ChatPromptTemplate.from_messages([
    ("system",
     "You are a Generative AI course study assistant.\n"
     "Use ONLY the provided course context to answer.\n"
     "If the answer is not in the context, say exactly: "
     "\"I don't know based on the course material.\"\n"
     "In this case, DO NOT include any source or page.\n"
     "If the answer IS found in the context:\n"
     "Be concise (max 6 lines).\n"
     "At the end of the answer, include a line starting with:\n"
     "Source: <file name> (page <page number>).\n"
     "Do NOT invent sources or page numbers.\n"
     "If possible, mention the relevant module or page in the answer.\n"
     "Do NOT invent facts or sources.\n"
     "Do NOT use outside knowledge.\n"
     "Do NOT guess.\n"
     ),
    ("human",
     "Question: {question}\n\n"
     "Course context:\n{context}\n\n"
     "Answer:")
])
```

Reliability Features

The chatbot includes the following reliability mechanisms:

- Answers are generated strictly from the retrieved course slides using RAG.
- If the answer is not found in the context, the model responds with: "I don't know based on the course material."
- Prompt injection attempts are mitigated through strict system instructions.
- Source citations (file name and page number) are included only when applicable.

These features reduce hallucinations and ensure trustworthy responses.

Step 8: Build LCEL RAG Chain

```
def format_docs(docs):
    out = []
    for d in docs:
        src = d.metadata.get("source", "unknown")
        page = d.metadata.get("page", "unknown")
        out.append(f"[source: {src} | page: {page}]\n{d.page_content}")
    return "\n\n---\n\n".join(out)


rag_chain = (
    {
        "context": retriever | format_docs,
        "question": RunnablePassthrough()
    }
    | rag_prompt
    | llm_used
    | StrOutputParser()
)

print(rag_chain.invoke("Who won the FIFA World Cup 2022?"))
```

```
 I don't know based on the course material.
```

Step 9: Test the Chatbot

```
questions = [
    "how is content creation a core characteristic of GenAI?",
    "what does the likelihood function represent?",
    "An encoder-only model consists only of what?",
    "what does zero-shot prompting mean?",
    "where is BPE used in?",
    "what is Distillation?",
    "what is Prompt management?",
    "what is Faiss?",
]

for q in questions:
    print("Q:", q)
    print("A:", rag_chain.invoke(q))
    print("-" * 60)
```

```
Q: how is content creation a core characteristic of GenAI?
A:  Content creation is a core characteristic of GenAI because it generates new content like essays, poems, artwork, music, and
Source: HTU - CPD - GenAI - Module1 (1).pdf (page 14).
------------------------------------------------------------
Q: what does the likelihood function represent?
A:  The likelihood function represents the probability of the observed data given specific model parameters. It is often transfo

Source: data/HTU - CPD - GenAI - Module2-A.pdf (page 17).
------------------------------------------------------------
Q: An encoder-only model consists only of what?
A:  An encoder-only model consists only of an encoder, which processes the input sequence and outputs contextualized vector embe

Source: HTU - CPD - GenAI - Module2-B.pdf (page 20).
------------------------------------------------------------
Q: what does zero-shot prompting mean?
A:  Zero-shot prompting involves giving the model a direct instruction to perform a task without any examples or demonstrations

Source: HTU - CPD - GenAI - Module3.pdf (page 22).
------------------------------------------------------------
Q: where is BPE used in?
A:  BPE is used in GPT-3.5, GPT-4, GPT-4o, and GPT-5.
Source: data/HTU - CPD - GenAI - Module4 (1).pdf (page 33).
```

```
------------------------------------------------------------
Q: what is Distillation?
A:  Distillation is the process of training a smaller model (student) to mimic a larger one (teacher). Core approaches include l

Source: HTU - CPD - GenAI - Module5.pdf (page 22).
------------------------------------------------------------
Q: what is Prompt management?
A:  I don't know based on the course material.
------------------------------------------------------------
Q: what is Faiss?
A:  Faiss is a library for efficient similarity search and clustering of dense vectors. It contains algorithms for searching in

Source: HTU - CPD - GenAI - Module6-B-RAG.pdf (page 22).
------------------------------------------------------------
```

Step 10: Evaluation – Plain LLM (Without RAG)

```
plain_prompt = ChatPromptTemplate.from_messages([
    ("system", "You are a Generative AI course assistant. Answer briefly."),
    ("human", "{question}")
])

plain_chain = plain_prompt | llm_used | StrOutputParser()
```

Step 11: Evaluation – With vs Without RAG

```
eval_questions = [
    "What is RAG?",
    "What does zero-shot prompting mean?",
    "What is distillation?"
]

for q in eval_questions:
    print("Q:", q)
    print("Without RAG:", plain_chain.invoke({"question": q}))
    print("With RAG:", rag_chain.invoke(q))
    print("-"*70)
```

```
Q: What is RAG?
Without RAG:
With RAG:  RAG (Retrieval-Augmented Generation) is an AI framework that combines information retrieval systems with generative l
Source: data/HTU - CPD - GenAI - Module6-B-RAG.pdf (page 5).
----------------------------------------------------------------------
Q: What does zero-shot prompting mean?
Without RAG:
With RAG:  Zero-shot prompting is a method where the prompt directly instructs the model to perform a task without providing any

Source: HTU - CPD - GenAI - Module3.pdf (page 22).
----------------------------------------------------------------------
Q: What is distillation?
Without RAG:  Distillation is a process used to separate components of a liquid mixture based on their boiling points. It involv
With RAG:  Distillation is the process of training a smaller model (student) to mimic a larger one (teacher). Core approaches in

Source: HTU - CPD - GenAI - Module5.pdf (page 22).
----------------------------------------------------------------------
```

Evaluation (With vs Without RAG)

We evaluated the chatbot using a manual comparison between responses generated with and without Retrieval-Augmented Generation (RAG).

Without RAG, the model produced generic or incorrect answers. For example, when asked about distillation, the model incorrectly explained it as a chemical process, which is unrelated to the course material.

With RAG enabled, the chatbot generated accurate and course-aligned answers, such as correctly describing distillation as training a smaller student model to mimic a larger teacher model.

This comparison demonstrates that RAG significantly improves answer accuracy and reduces hallucinations by grounding responses in the provided course slides.

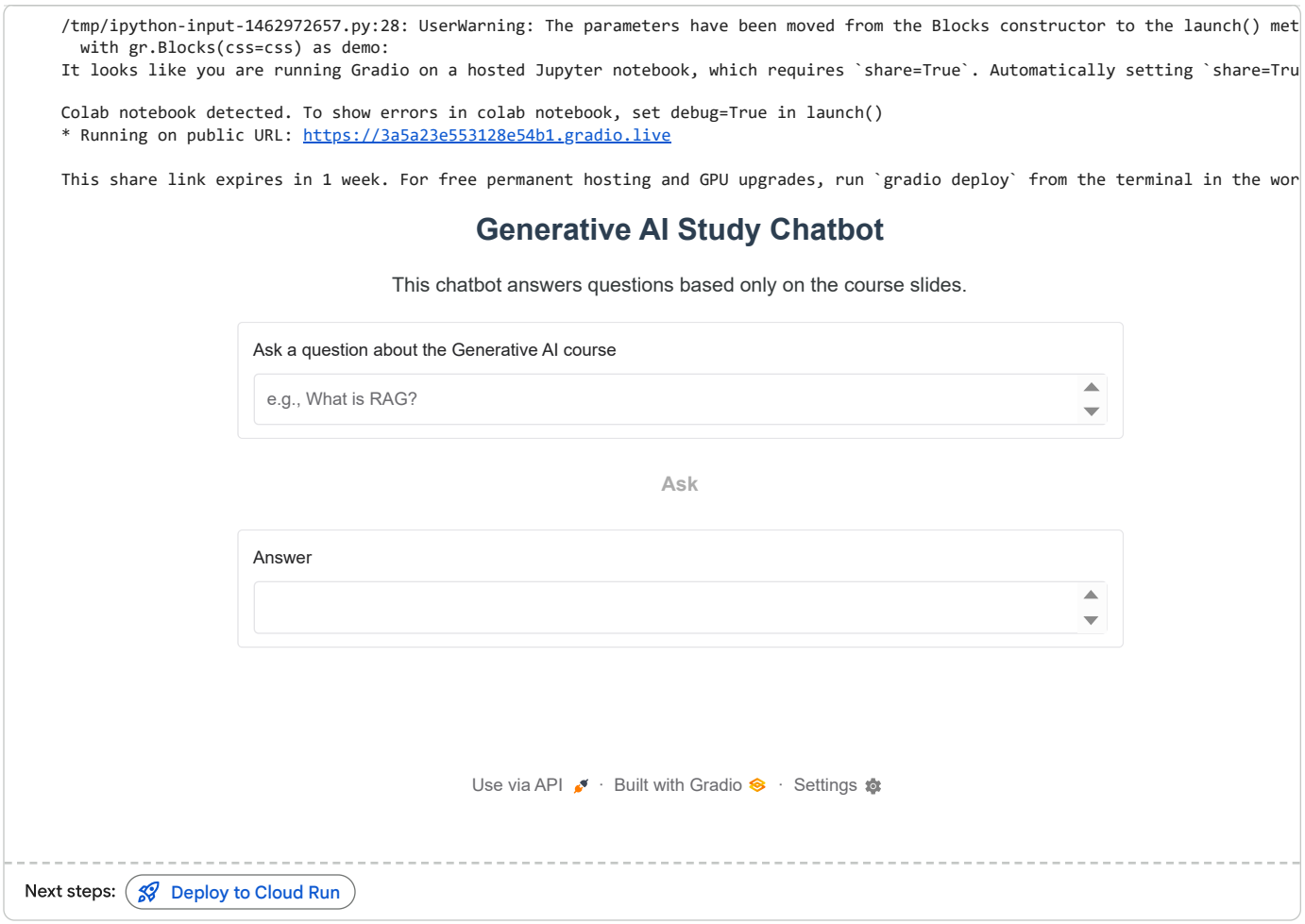
Step 12: Gradio GUI

Chat Memory Note

The chat memory is used only for displaying recent conversation history to the user interface. Each question is processed independently by the RAG pipeline to avoid context leakage and hallucination.

```
import gradio as gr

chat_history = []

def chatbot(question):
    if not question.strip():
        return "Please enter a valid question."

    chat_history.append(f"User: {question}")
    answer = rag_chain.invoke(question)
    chat_history.append(f"Bot: {answer}")

    # Kept the last 6 messages only.
    recent_history = "\n".join(chat_history[-6:])

    return recent_history


# CSS for styling
css = """
.gradio-container { font-family: 'Arial', sans-serif; }
.title { text-align: center; font-size: 24px; color: #2c3e50; }
.description { text-align: center; font-size: 16px; color: #7f8c8d; }
.textbox { font-size: 16px; border-radius: 10px; }
.button { background-color: #3498db; color: white; border: none; border-radius: 10px; }
"""

with gr.Blocks(css=css) as demo:
    gr.Markdown("<h1 class='title'>Generative AI Study Chatbot</h1>", elem_classes="title")
    gr.Markdown(
        "<p class='description'>This chatbot answers questions based only on the course slides.</p>",
        elem_classes="description"
    )

    with gr.Row():
        textbox = gr.Textbox(
            label="Ask a question about the Generative AI course",
            placeholder="e.g., What is RAG?",
            elem_classes="textbox"
        )

    with gr.Row():
        submit_btn = gr.Button("Ask", elem_classes="button")

    with gr.Row():
        output = gr.Textbox(label="Answer", elem_classes="textbox")

    submit_btn.click(
        fn=chatbot,
        inputs=textbox,
        outputs=output
    )

demo.launch()
```

```
/tmp/ipython-input-1462972657.py:28: UserWarning: The parameters have been moved from the Blocks constructor to the launch() met
  with gr.Blocks(css=css) as demo:
It looks like you are running Gradio on a hosted Jupyter notebook, which requires `share=True`. Automatically setting `share=Tru

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://3a5a23e553128e54b1.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the wor
```



Reflections:

The chatbot successfully answered course-related questions using RAG with the provided slides. Using FAISS and embeddings significantly improved answer accuracy compared to a plain LLM. The main challenge was handling large slide content and choosing suitable chunk sizes. RAG reduced hallucinations by grounding answers in real course material. Adding a strict prompt improved reliability by preventing out-of-context answers. The Gradio interface made interaction easier and more user-friendly.