''' Here chatBot implemented is basically for taking the user input in the form of .PDF , .txt , .csv format. Used Hugging Face Sentence transformer , used tiiuae/falcon-7b-instruct which is free in nature.

PipeLine*

1.Document Ingestion (Input Layer)

2. Text Preprocessing & Chunking
3. Embedding Generation (Retriever Side)
4. Vector Database
5. Retriever (Search Engine)
6. LLM (Generator Side)
7. Response Delivery (Chat Interface) '''

```
pip install langchain chromadb sentence-transformers transformers pypdf pandas
```

Show hidden output

```
pip install -U langchain-community
```

Show hidden output

```
pip install -U langchain langchain-community
```

Show hidden output

```
# Loading the basic libraries
import os
from langchain.document_loaders import PyPDFLoader, TextLoader, CSVLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import Chroma
from langchain.chains import RetrievalQA
from langchain.llms import HuggingFacePipeline
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
```

1. Document Ingestion (Input Layer)

The user uploads files (PDF, TXT, CSV).

To extract raw text for further processing.

Tools used: PyPDFLoader, TextLoader, CSVLoader (LangChain document loaders).

```
# 1. Load multiple documents
def load_documents(file_paths):
    all_docs = []
    for file_path in file_paths:
        if file_path.endswith('.pdf'):
            loader = PyPDFLoader(file_path)
        elif file_path.endswith('.txt'):
            loader = TextLoader(file_path)
        elif file_path.endswith('.csv'):
            loader = CSVLoader(file_path)
        else:
            raise ValueError(f"Unsupported file format: {file_path}")

        all_docs.extend(loader.load())
    return all_docs
```

2. Text Preprocessing & Chunking What happens? Large text is split into chunks (e.g., 500–1000 tokens) with some overlap.

Why? LLMs have context window limits (e.g., 4k tokens). Chunking keeps context relevant and efficient.

Tools used: RecursiveCharacterTextSplitter (LangChain).

NLP role: Breaking text at logical points, preserving meaning with overlap.

```
# 2. Split documents into chunks
def split_documents(documents, chunk_size=1000, chunk_overlap=200):
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=chunk_size,
```

```
        chunk_overlap=chunk_overlap
    )
    return text_splitter.split_documents(documents)
```

3. Embedding Generation (Retriever Side) What happens? Each chunk → converted into dense vector representation.

How? Using a Transformer-based encoder model (e.g., sentence-transformers/all-MiniLM-L6-v2).

Why? To enable semantic search—finding chunks similar to the query.

Tools used: HuggingFaceEmbeddings (LangChain).

Transformer role: ✓ Encoder-only model (like BERT/MiniLM) is used here. ✓ It uses self-attention to capture semantic meaning of words in context.

```
# 3. Create embeddings using HuggingFace
def create_vector_store(docs):
    embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
    vectorstore = Chroma.from_documents(docs, embedding_model)
    return vectorstore
```

4. Vector Database What happens? Store all embeddings in a vector store for fast similarity search.

Why? Efficient retrieval using cosine similarity or dot product.

Tools used: Chroma or FAISS.

NLP role: None here—this is pure vector storage and retrieval.

```
# 4. Setup Free HuggingFace LLM
def get_llm():
    model_name = "tiiuae/falcon-7b-instruct"  # Or "gpt2" for faster testing
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForCausalLM.from_pretrained(model_name)

    hf_pipeline = pipeline(
        "text-generation",
        model=model,
        tokenizer=tokenizer,
        max_new_tokens=512,
        temperature=0.2
    )

    return HuggingFacePipeline(pipeline=hf_pipeline)
```

5. Retriever (Search Engine) What happens? When the user asks a question, convert query → embedding → retrieve top-k most relevant chunks.

Why? Provide the LLM with grounded context from original documents.

Tools used: vectorstore.as_retriever() in LangChain.

Transformer role: Uses the same embedding model to encode the query.

```
# 5. Build RAG Pipeline
def build_rag_pipeline(vectorstore):
    llm = get_llm()
    retriever = vectorstore.as_retriever(search_kwargs={"k": 3})
    qa_chain = RetrievalQA.from_chain_type(
        llm=llm,
        retriever=retriever,
        chain_type="stuff"
    )
    return qa_chain
```

6. LLM (Generator Side) What happens? The retrieved chunks + user query → passed to the Language Model.

Why? To generate a natural language response using both context and query.

Tools used: Hugging Face models (e.g., tiiuae/falcon-7b-instruct) loaded with transformers pipeline.

Transformer role: ✓ Decoder-only Transformer generates the final answer token by token. ✓ Uses self-attention + causal masking for autoregressive generation.

```
# 6. Main Chatbot
def rag_chatbot(file_paths):
    print("Loading documents...")
    documents = load_documents(file_paths)
```

```
    print("Splitting documents...")
    docs = split_documents(documents)

    print("Creating vector store...")
    vectorstore = create_vector_store(docs)

    print("Building RAG pipeline...")
    qa_chain = build_rag_pipeline(vectorstore)

    print("\nChatbot is ready! Type 'exit' to quit.\n")

    while True:
        query = input("You: ")
        if query.lower() == "exit":
            break
        response = qa_chain.run(query)
        print("Bot:", response)
```

7. Response Delivery What happens? The LLM outputs the answer to the user.

NLP role: Final text generation and formatting.

```
# Run the chatbot
if __name__ == "__main__":
    file_paths = ["/content/sample_data/RAG_Chatbot_Transformer_Complete_Guide.pdf"]  # Add your files here



rag_chatbot(file_paths)
```

```
Loading documents...
Splitting documents...
Creating vector store...
/tmp/ipython-input-490645480.py:3: LangChainDeprecationWarning: The class `HuggingFaceEmbeddings` was deprecated in LangChain 0.2.2
  embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
/usr/local/lib/python3.11/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 as s
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, 45.4kB/s]
config_sentence_transformers.json: 100%                             116/116 [00:00<00:00, 17.2kB/s]
README.md:         10.5k/? [00:00<00:00, 1.43MB/s]
sentence_bert_config.json: 100%                                     53.0/53.0 [00:00<00:00, 8.19kB/s]
config.json: 100%                                                   612/612 [00:00<00:00, 92.1kB/s]
model.safetensors: 100%                                             90.9M/90.9M [00:01<00:00, 76.0MB/s]
tokenizer_config.json: 100%                                         350/350 [00:00<00:00, 50.8kB/s]
vocab.txt:         232k/? [00:00<00:00, 7.14MB/s]
tokenizer.json:         466k/? [00:00<00:00, 24.1MB/s]
special_tokens_map.json: 100%                                       112/112 [00:00<00:00, 16.3kB/s]
config.json: 100%                                                   190/190 [00:00<00:00, 25.4kB/s]
/usr/local/lib/python3.11/dist-packages/torch/nn/modules/module.py:1750: FutureWarning: `encoder_attention_mask` is deprecated and v
  return forward_call(*args, **kwargs)
Building RAG pipeline...
tokenizer_config.json:         1.13k/? [00:00<00:00, 149kB/s]
tokenizer.json:         2.73M/? [00:00<00:00, 105MB/s]
special_tokens_map.json: 100%                                       281/281 [00:00<00:00, 42.1kB/s]
config.json:         1.05k/? [00:00<00:00, 141kB/s]
model.safetensors.index.json:         17.7k/? [00:00<00:00, 2.28MB/s]
Fetching 2 files: 100%                                              2/2 [01:28<00:00, 88.41s/it]
model-00001-of-00002.safetensors: 100%                              9.95G/9.95G [01:28<00:00, 215MB/s]
model-00002-of-00002.safetensors: 100%                              4.48G/4.48G [00:20<00:00, 24.6MB/s]
Loading checkpoint shards: 100%                                     2/2 [00:27<00:00, 12.72s/it]
generation_config.json: 100%                                        117/117 [00:00<00:00, 18.0kB/s]
Device set to use cpu
/tmp/ipython-input-2498918634.py:15: LangChainDeprecationWarning: The class `HuggingFacePipeline` was deprecated in LangChain 0.0.37
  return HuggingFacePipeline(pipeline=hf_pipeline)

Chatbot is ready! Type 'exit' to quit.

You: what are tokens
/tmp/ipython-input-1931676148.py:21: LangChainDeprecationWarning: The method `Chain.run` was deprecated in langchain 0.1.0 and will
  response = qa_chain.run(query)
/usr/local/lib/python3.11/dist-packages/torch/nn/modules/module.py:1750: FutureWarning: `encoder_attention_mask` is deprecated and v
  return forward_call(*args, **kwargs)
Setting `pad_token_id` to `eos_token_id`:11 for open-end generation.
Bot: Use the following pieces of context to answer the question at the end. If you don't know the answer, just say that you don't kn

✓ Mention tokenization and positional encoding as essential for context.
Pro Tip: If asked, draw the Transformer block (attention + feed-forward) and explain its role in
retrieval and generation.

✓ Attention Mechanism: Core concept that lets models focus on important parts of the input
sequence.
✓ Encoder: Used in embedding models (retrieval). Converts chunks into contextual embeddings.
✓ Decoder: Used in LLM (generation). Takes user query + retrieved context and predicts next
tokens.
Key Components:
- Self-Attention: Calculates attention weights between tokens in the same sequence.
- Multi-Head Attention: Captures multiple relationships simultaneously.
- Feed Forward Network: Adds non-linearity for better representation.
- Positional Encoding: Adds order info to sequences.
How Encoder-Decoder Works for RAG:
✓ Retriever uses an Encoder-only Transformer (e.g., BERT) to create embeddings of text
chunks.
✓ Generator uses a Decoder-only Transformer (e.g., GPT) to generate responses.
✓ RAG pipeline:
- User query → encoded → retrieve top-k chunks from vector DB.
- Concatenate context + query → fed into LLM (Transformer-based).
- Model outputs generated answer.
Visual Flow:
```

```
RAG Chatbot + Transformer Architecture -
 Complete Guide
What is RAG?
Retrieval-Augmented Generation (RAG) combines:
✓ Retrieval: Fetches relevant data chunks from a knowledge base.
✓ Generation: LLM creates the final answer using the retrieved context.
Why use RAG?
✓ Reduces hallucinations by grounding answers in factual data.
✓ Cost-effective compared to full LLM fine-tuning.
✓ Dynamic: Easily update knowledge without retraining the model.
Role of Transformers in RAG:
Transformers are used in two places:
1. Retriever: Embedding models like BERT or MiniLM (encoder-based transformers) convert text
chunks into vectors.
2. Generator: Language Models like Falcon, GPT, LLaMA (decoder-only transformers) generate
responses from retrieved context.
Transformer Architecture in Context of RAG:
✓ Attention Mechanism: Core concept that lets models focus on important parts of the input
sequence.
✓ Encoder: Used in embedding models (retrieval). Converts chunks into contextual embeddings.

Question: what are tokens
Helpful Answer:
Tokens are small units of text that are used to represent words, phrases, or sentences in natural language. They are typically repre
You: provide 10 important points to remember from the document
/usr/local/lib/python3.11/dist-packages/torch/nn/modules/module.py:1750: FutureWarning: `encoder_attention_mask` is deprecated and w
  return forward_call(*args, **kwargs)
Setting `pad_token_id` to `eos_token_id`:11 for open-end generation.
Bot: Use the following pieces of context to answer the question at the end. If you don't know the answer, just say that you don't kr

✓ Mention tokenization and positional encoding as essential for context.
Pro Tip: If asked, draw the Transformer block (attention + feed-forward) and explain its role in
retrieval and generation.

✓ Attention Mechanism: Core concept that lets models focus on important parts of the input
sequence.
✓ Encoder: Used in embedding models (retrieval). Converts chunks into contextual embeddings.
✓ Decoder: Used in LLM (generation). Takes user query + retrieved context and predicts next
tokens.
Key Components:
- Self-Attention: Calculates attention weights between tokens in the same sequence.
- Multi-Head Attention: Captures multiple relationships simultaneously.
- Feed Forward Network: Adds non-linearity for better representation.
- Positional Encoding: Adds order info to sequences.
How Encoder-Decoder Works for RAG:
✓ Retriever uses an Encoder-only Transformer (e.g., BERT) to create embeddings of text
chunks.
✓ Generator uses a Decoder-only Transformer (e.g., GPT) to generate responses.
✓ RAG pipeline:
- User query → encoded → retrieve top-k chunks from vector DB.
- Concatenate context + query → fed into LLM (Transformer-based).
- Model outputs generated answer.
Visual Flow:

RAG Chatbot + Transformer Architecture -
 Complete Guide
What is RAG?
Retrieval-Augmented Generation (RAG) combines:
✓ Retrieval: Fetches relevant data chunks from a knowledge base.
✓ Generation: LLM creates the final answer using the retrieved context.
Why use RAG?
✓ Reduces hallucinations by grounding answers in factual data.
✓ Cost-effective compared to full LLM fine-tuning.
✓ Dynamic: Easily update knowledge without retraining the model.
Role of Transformers in RAG:
Transformers are used in two places:
1. Retriever: Embedding models like BERT or MiniLM (encoder-based transformers) convert text
chunks into vectors.
2. Generator: Language Models like Falcon, GPT, LLaMA (decoder-only transformers) generate
responses from retrieved context.
Transformer Architecture in Context of RAG:
✓ Attention Mechanism: Core concept that lets models focus on important parts of the input
sequence.
✓ Encoder: Used in embedding models (retrieval). Converts chunks into contextual embeddings.

Question: provide 10 important points to remember from the document
Helpful Answer:
✓ RAG is a combination of Retrieval and Generation.
✓ Retrieval is the process of finding relevant data chunks from a knowledge base.
✓ Generation is the process of creating a response from the retrieved context.
✓ RAG is a cost-effective way to fine-tune a model for generation.
✓ RAG is dynamic and can be updated without retraining the model.
✓ Transformers are used in two places: Retriever and Generator.
✓ Transformers are used in RAG to create embeddings of text chunks.
✓ Transformers are used in RAG to generate responses from retrieved context.
✓ Retrieval is the process of finding relevant data chunks from a knowledge base.
✓ Generation is the process of creating a response from the retrieved context.
✓ RAG is a cost-effective way to fine-tune a model for generation.
✓ RAG is dynamic and can be updated without retraining the model.
✓ Transformers are used in two places: Retriever and Generator.
```

```
✓ Transformers are used in RAG to create embeddings of text chunks.
✓ Transformers are used in RAG to generate responses from retrieved context.
✓ Retrieval is the process of finding relevant data chunks from a knowledge base.
✓ Generation is the process of creating a response from the retrieved context.
✓ RAG is a cost-effective way to fine-tune a model for generation.
✓ RAG is dynamic and can be updated without retraining the model.
✓ Transformers are used in two places: Retriever and Generator.
✓ Transformers are used in RAG to create embeddings of text chunks.
✓ Transformers are used in RAG to generate responses from retrieved context.
✓ Retrieval is the process of finding relevant data chunks from a knowledge base.
✓ Generation is the process of creating a response from the retrieved context.
✓ RAG is a cost-effective way to fine-tune a model for generation.
✓ RAG is dynamic and can be updated without retraining the model.
✓ Transformers are used in two places: Retriever and Generator.
✓ Transformers are used in RAG to create embeddings of text chunks.
✓ Transformers are used in RAG to generate responses from retrieved context.
✓ Retrieval is the process of finding relevant data chunks from a knowledge base.
✓ Generation is the process of creating a response from the retrieved context.
✓ RAG is a cost
You: whi is Nitin
/usr/local/lib/python3.11/dist-packages/torch/nn/modules/module.py:1750: FutureWarning: `encoder_attention_mask` is deprecated and w
  return forward_call(*args, **kwargs)
Setting `pad_token_id` to `eos_token_id`:11 for open-end generation.
Bot: Use the following pieces of context to answer the question at the end. If you don't know the answer, just say that you don't kr

✓ Attention Mechanism: Core concept that lets models focus on important parts of the input
sequence.
✓ Encoder: Used in embedding models (retrieval). Converts chunks into contextual embeddings.
✓ Decoder: Used in LLM (generation). Takes user query + retrieved context and predicts next
tokens.
Key Components:
- Self-Attention: Calculates attention weights between tokens in the same sequence.
- Multi-Head Attention: Captures multiple relationships simultaneously.
- Feed Forward Network: Adds non-linearity for better representation.
- Positional Encoding: Adds order info to sequences.
How Encoder-Decoder Works for RAG:
✓ Retriever uses an Encoder-only Transformer (e.g., BERT) to create embeddings of text
chunks.
✓ Generator uses a Decoder-only Transformer (e.g., GPT) to generate responses.
✓ RAG pipeline:
- User query → encoded → retrieve top-k chunks from vector DB.
- Concatenate context + query → fed into LLM (Transformer-based).
- Model outputs generated answer.
Visual Flow:

✓ RAG pipeline:
- User query → encoded → retrieve top-k chunks from vector DB.
- Concatenate context + query → fed into LLM (Transformer-based).
- Model outputs generated answer.
Visual Flow:
User Query → [Retriever (Encoder-based Transformer)] → Relevant Chunks → [Generator
(Decoder Transformer)] → Answer.
Key Points to Impress Interviewers:
✓ Explain that embeddings come from encoder Transformers.
✓ LLM generation is handled by decoder-only Transformers.
✓ Attention mechanism is why Transformers outperform RNNs.
✓ Multi-head attention captures multiple semantic relations.

RAG Chatbot + Transformer Architecture -
 Complete Guide
What is RAG?
Retrieval-Augmented Generation (RAG) combines:
✓ Retrieval: Fetches relevant data chunks from a knowledge base.
✓ Generation: LLM creates the final answer using the retrieved context.
Why use RAG?
✓ Reduces hallucinations by grounding answers in factual data.
✓ Cost-effective compared to full LLM fine-tuning.
✓ Dynamic: Easily update knowledge without retraining the model.
Role of Transformers in RAG:
Transformers are used in two places:
1. Retriever: Embedding models like BERT or MiniLM (encoder-based transformers) convert text
chunks into vectors.
2. Generator: Language Models like Falcon, GPT, LLaMA (decoder-only transformers) generate
responses from retrieved context.
Transformer Architecture in Context of RAG:
✓ Attention Mechanism: Core concept that lets models focus on important parts of the input
sequence.
✓ Encoder: Used in embedding models (retrieval). Converts chunks into contextual embeddings.

Question: whi is Nitin
Helpful Answer: Nitin is a language model that is used in natural language processing. It is used to generate responses to user quer
You: exit
```

RAG(Retrieval-Augmented Generation) based Chat Bot Prompt.