CODE

from transformers import pipeline

from langchain.document\_loaders import TextLoader

from langchain.text\_splitter import RecursiveCharacterTextSplitter

from langchain.chains import RetrievalQA

from langchain.embeddings.openai import OpenAIEmbeddings

from langchain.vectorstores import FAISS

# Step 1: Load the document

def load\_document(file\_path):

with open(file\_path, 'r') as file:

return file.read()

# Step 2: Split the document into smaller chunks

def split\_document(text):

text\_splitter = RecursiveCharacterTextSplitter(

chunk\_size=1000, # Customize based on your document size

chunk\_overlap=200

)

return text\_splitter.split\_text(text)

# Step 3: Create a vector store for retrieval

def create\_retriever(chunks):

embeddings = OpenAIEmbeddings() # Replace with other embeddings if needed

vector\_store = FAISS.from\_texts(chunks, embeddings)

return vector\_store

# Step 4: Build a QA pipeline

def build\_qa\_pipeline(vector\_store):

retriever = vector\_store.as\_retriever()

qa\_pipeline = RetrievalQA.from\_chain\_type(

retriever=retriever,

chain\_type="stuff", # Default method of chaining

llm=pipeline("text-davinci-003") # Replace with an alternative model

)

return qa\_pipeline

# Main function

def main():

# Path to your document

file\_path = "document.txt" # Replace with your document's path

# Load the document

document = load\_document(file\_path)

# Split the document into chunks

chunks = split\_document(document)

# Create a retriever

vector\_store = create\_retriever(chunks)

# Build the QA pipeline

qa\_pipeline = build\_qa\_pipeline(vector\_store)

print("Ask a question (type 'exit' to quit):")

while True:

question = input("\nYour Question: ")

if question.lower() == "exit":

break

answer = qa\_pipeline.run(question)

print(f"Answer: {answer}")

if \_\_name\_\_ == "\_\_main\_\_":

main()