- **Introduction:**

This script implements a simple Retrieval-Augmented Generation pipeline. It either loads or builds a FAISS vector index using langchain from an edge computing text corpus **file name - edge_computing_cleaned.txt**, it retrieves the most relevant chunks (retrieving top 5 records) for a user question, and feeds those as context to a local large language model to produce a concise answer. To use the ollama, I need to install it on my local machine so that I can access it and can use ChatOllama to retrieve the data from file.

Fig1.1 and Fig1.2 Shows the main implementation of the RAG using langchain that shows how we upload the file, after that build a FAISS vector index to retrieve the relevant data and **Fig2**., shows how we are calling the main file and what information we want.

To run this code, you must use - **python run_research_assistant.py** command (as shown in output) this will call **research_assistant.py** file that contains runnable implementations.

- **Code Screenshot:**

research_assistant.py | run_research_assistant.py

research_assistant.py > answer_question

```python
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import FAISS
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_ollama import ChatOllama
import os

# Initialize embeddings model (needed for either loading or building index)
embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
index_dir = "faiss_new_index"
if os.path.isdir(index_dir):
    print(f"FAISS index directory '{index_dir}' exists. Loading existing index; skipping text processing.")
    faiss_index = FAISS.load_local(
        index_dir,
        embedding_model,
        allow_dangerous_deserialization=True
    )
else:
    print(f"FAISS index directory '{index_dir}' not found. Building new index...")
    # Load and split only when index absent
    file_path = "edge_computing_cleaned.txt"
    with open(file_path, "r", encoding="utf-8") as f:
        text = f.read()

    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=500,
        chunk_overlap=50,
        separators=["\n\n", "\n", ".", " ", ""]
    )
    chunks = text_splitter.split_text(text)
    print(f"Total chunks: {len(chunks)}\n")
    for i, chunk in enumerate(chunks[:5]):
        print(f"--- Chunk {i+1} ---")
        print(chunk)
        print()

    faiss_index = FAISS.from_texts(chunks, embedding_model)
    faiss_index.save_local(index_dir)
    print(f"FAISS index saved to '{index_dir}' directory.")

llm = ChatOllama(model="llama3")   # your local Llama-3 instance

def answer_question(question, k=5):
```

Fig1.1: Implementation Code

```python
def answer_question(question, k=5):
    docs = faiss_index.similarity_search(question, k=k)
    context = "\n\n".join([d.page_content for d in docs])
    prompt = (
        "Use the following context to answer concisely:\n"
        f"{context}\n\n"
        f"Question: {question}\n"
        "Answer:"
    )
    response = llm.invoke(prompt)
    return response.content
```

Fig1.2: Implementation Code

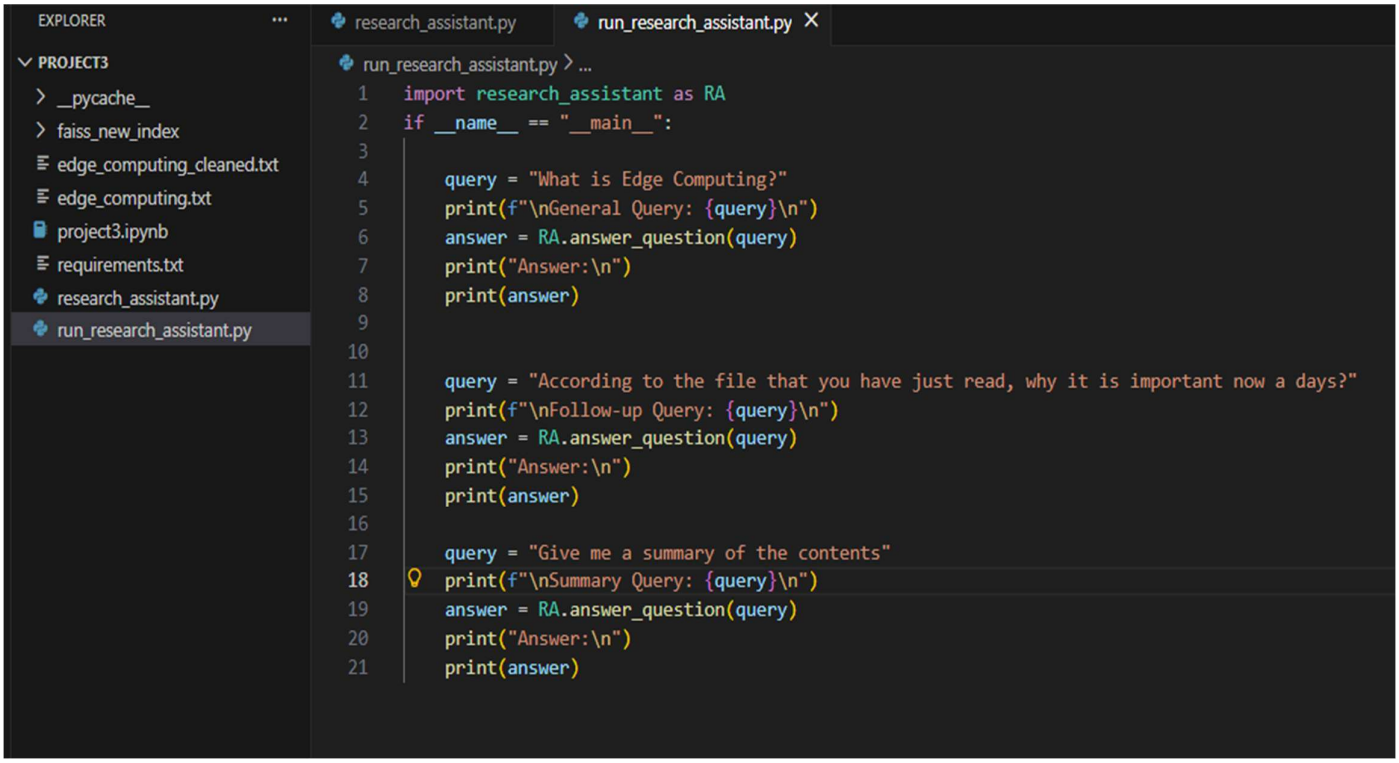

Fig2: Demonstrating Usage

- **Output:**

```
PS C:\Users\achop\Documents\CIS583\project3> python .\run_research_assistant.py
2025-11-21 12:55:09.454366: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different c
omputation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
WARNING:tensorflow:From C:\Users\achop\AppData\Local\Programs\Python\Python310\lib\site-packages\tf_keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use t
f.compat.v1.losses.sparse_softmax_cross_entropy instead.

FAISS index directory 'faiss_new_index' exists. Loading existing index; skipping text processing.

General Query: What is Edge Computing?

Answer:

Edge computing is a way of processing data closer to where it's created, such as on phones or machines, instead of sending it all the way to a distant cloud or data center. This reduces delay, cuts dow
n on internet usage, and enables faster real-time responses.

Follow-up Query: According to the file that you have just read, why it is important now a days?

Answer:

According to the text, edge computing is important nowadays because it enables devices to respond faster and process information in real-time, even in places with slow or unavailable cloud connections.
 This prevents network congestion, reduces latency, and increases data privacy, which is crucial for applications like self-driving cars and smart cities where rapid responses are critical.

Summary Query: Give me a summary of the contents

Answer:

The context discusses examples of AI agents in research papers, including:

* Sensors tracking soil moisture, temperature, and crop conditions, sending only critical updates to trigger irrigation systems.
* Trains equipped with sensors detecting mechanical faults or vibrations, analyzing data onboard and sending important alerts or maintenance requests.

In summary, the contents showcase practical applications of AI agents in monitoring and control systems.
PS C:\Users\achop\Documents\CIS583\project3>
```

- **How does the attention + retrieval mechanism in Transformers compare to the sequence-based memory of RNNs/LSTMs?**

Sequential Based (RNN / LSTMs):

1. RNNs process inputs one step at a time in sequence.
2. They store memory in a hidden state passed from step to step.
3. LSTMs add gates to better control what to remember or forget.
4. Memory is compressed, so older information fades over time.
5. Training is sequential and slow, since each step depends on the previous one.

Transformers (Attention + retrieval):

1. Transformers process all tokens in parallel instead of step-by-step.
2. Attention compares every token with every other token to find relevance.
3. Memory is not a single vector the model can access the entire sequence directly.
4. Long-range dependencies are easy because any token can look anywhere.
5. Training is fast due to full parallelization and efficient gradient flow.
6. With retrieval augmentation (RAG), Transformers gain external memory far beyond the model size.

RNNs remember by carrying a compressed hidden state through time, while Transformers remember by directly looking back at any token using attention.