

# Local Research Assistant using Langchain + Ollama

## CIS 483/583 – Deep Learning

### Project 3

#### 1. Introduction:

This project aimed to create a fully local research assistant that can load text documents, break them into manageable parts, retrieve relevant information with vector search, and generate context-aware answers using a Large Language Model (LLaMA 3) running locally through Ollama.

Modern LLMs are powerful, but they have fixed context windows. Retrieval-Augmented Generation (RAG) offers a practical solution by embedding external documents and retrieving only the most relevant sections during queries. This allows the model to operate as if it has a much larger context window.

In this project, I set up a complete RAG pipeline using LangChain components like RecursiveCharacterTextSplitter, HuggingFaceEmbeddings, FAISS for vector storage, and ChatOllama for inference. After loading and indexing a collection of documents about AI, neural networks, and healthcare applications, the system was tested on factual, contextual, and summarization tasks. The result is a self-contained research assistant that runs entirely on a local machine without any cloud dependencies.

#### 2. System Design and Implementation

##### 2.1 Windows Environment Setup

Create and activate a virtual environment

```
python -m venv venv
```

Activate the virtual environment

```
venv/Scripts/activate
```

##### 2.2 Ollama Installation

Install Ollama if not already installed

Go to <https://ollama.com/download>

Download the installer

Test to make sure it is running

Go into the VS Code Terminal

*ollama list*

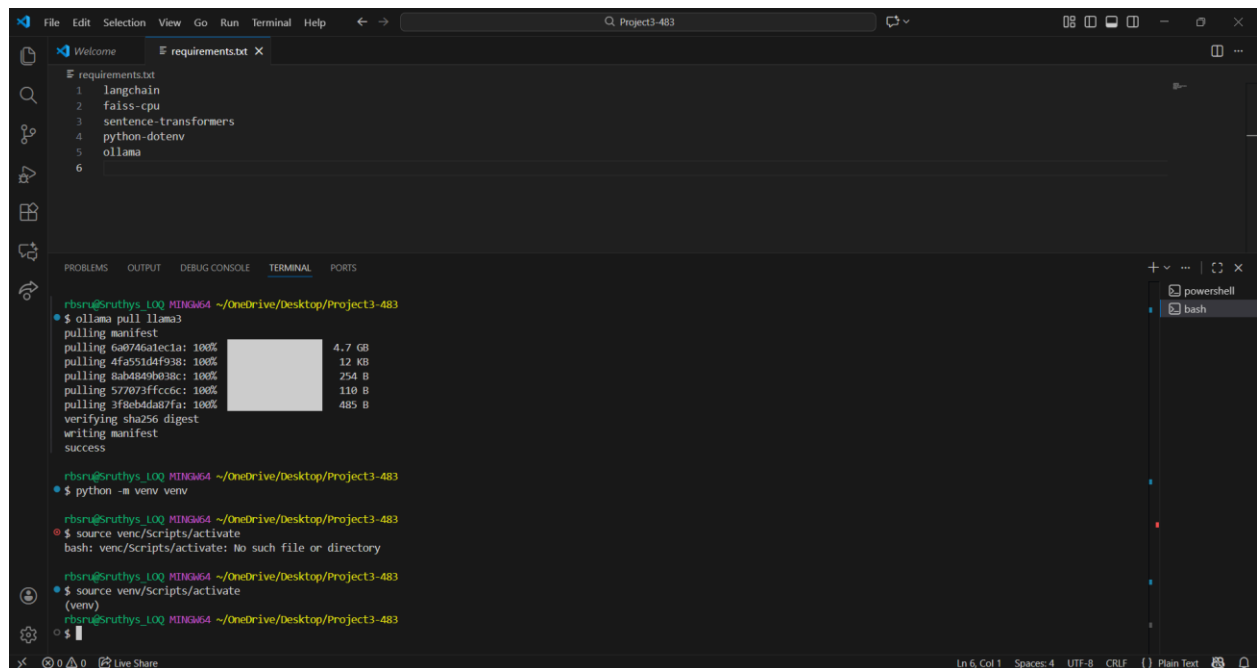
*ollama pull llama3*

Pull the embedding model: *ollama pull nomic-embed-text*

## 2.3 Dependencies

Install required packages

*pip install langchain langchain-community langchain-core langchain-ollama faiss-cpu*  
*pip install langchain langchain-community langchain-ollama langchain-text-splitters faiss-cpu*



```
File Edit Selection View Go Run Terminal Help
Project3-483

requirements.txt
1 langchain
2 faiss-cpu
3 sentence-transformers
4 python-dotenv
5 ollama
6

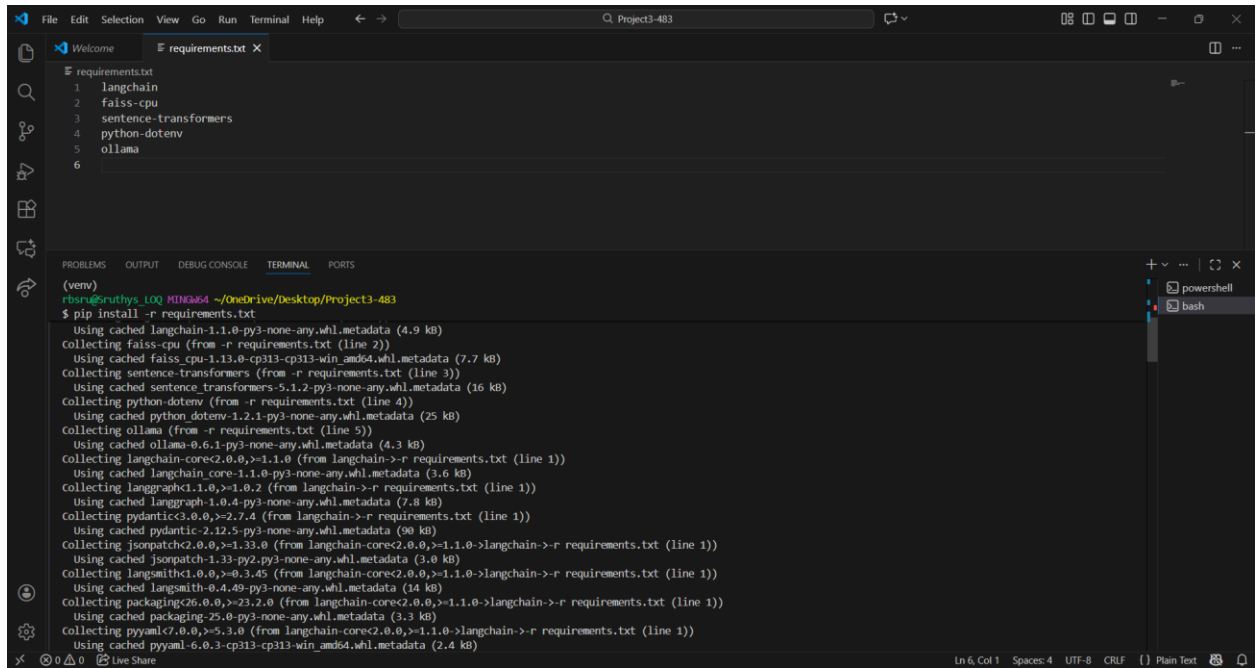
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

rbsru@sruthys_10Q MINGW64 ~/OneDrive/Desktop/Project3-483
$ ollama pull llama3
pulling manifest
pulling 6a0746a1ec1a: 100% 4.7 GB
pulling 4fa551d4f938: 100% 12 KB
pulling 8ab4849b038c: 100% 254 B
pulling 57073ffcc0c: 100% 110 B
pulling 3f0eb4da07fa: 100% 485 B
verifying sha256 digest
writing manifest
success

rbsru@sruthys_10Q MINGW64 ~/OneDrive/Desktop/Project3-483
$ python -m venv venv

rbsru@sruthys_10Q MINGW64 ~/OneDrive/Desktop/Project3-483
$ source venv/Scripts/activate
bash: venv/Scripts/activate: No such file or directory

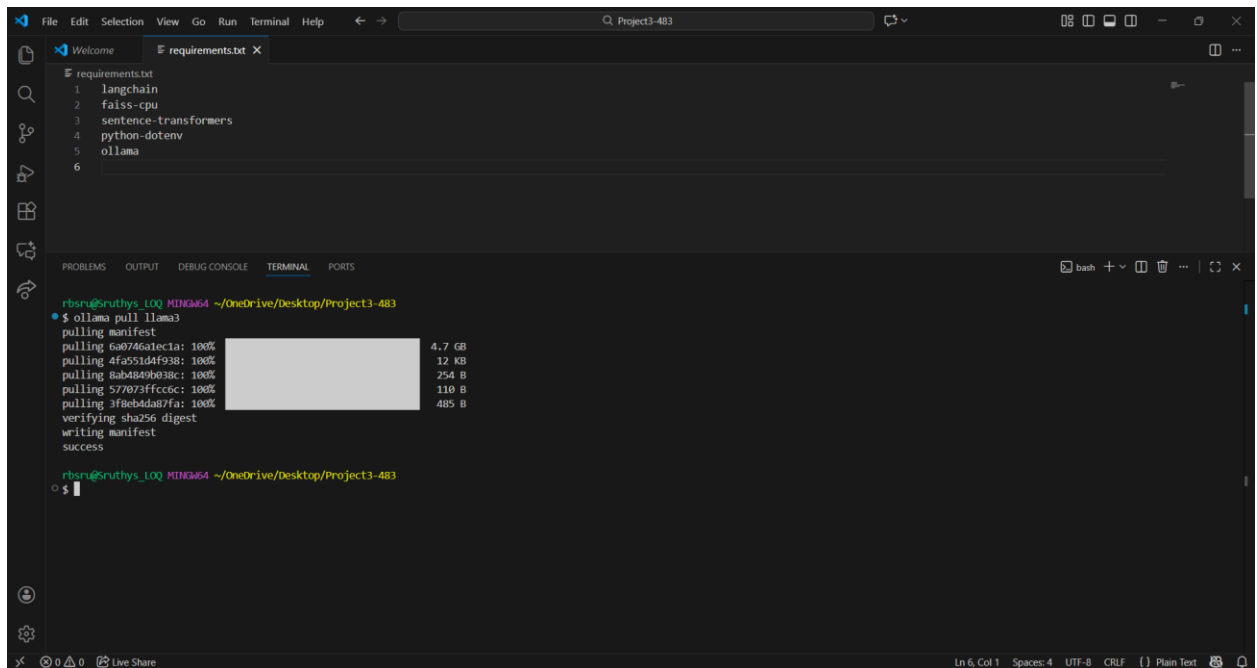
rbsru@sruthys_10Q MINGW64 ~/OneDrive/Desktop/Project3-483
$ source venv/Scripts/activate
(venv)
rbsru@sruthys_10Q MINGW64 ~/OneDrive/Desktop/Project3-483
$
```



The screenshot shows the Visual Studio Code editor with a file named `requirements.txt` open. The file contains the following dependencies: `langchain`, `faiss-cpu`, `sentence-transformers`, `python-dotenv`, and `ollama`. The terminal window at the bottom shows the command `$ pip install -r requirements.txt` being executed. The output displays the progress of installing each dependency, including the version and size of the files being downloaded.

```
requirements.txt
1 langchain
2 faiss-cpu
3 sentence-transformers
4 python-dotenv
5 ollama
6

(venv)
rbsru@sruthys_L0Q MINGW64 ~/OneDrive/Desktop/Project3-483
$ pip install -r requirements.txt
Using cached langchain-1.1.0-py3-none-any.whl.metadata (4.9 kB)
Collecting langchain (from -r requirements.txt (line 1))
  Using cached langchain-1.1.0-py3-none-any.whl.metadata (4.9 kB)
Collecting faiss-cpu (from -r requirements.txt (line 2))
  Using cached faiss_cpu-1.13.0-cp313-cp313-win_amd64.whl.metadata (7.7 kB)
Collecting sentence-transformers (from -r requirements.txt (line 3))
  Using cached sentence_transformers-5.1.2-py3-none-any.whl.metadata (16 kB)
Collecting python-dotenv (from -r requirements.txt (line 4))
  Using cached python_dotenv-1.2.1-py3-none-any.whl.metadata (25 kB)
Collecting ollama (from -r requirements.txt (line 5))
  Using cached ollama-0.6.1-py3-none-any.whl.metadata (4.3 kB)
Collecting langchain-core<2.0.0,>=1.1.0 (from langchain->-r requirements.txt (line 1))
  Using cached langchain_core-1.1.0-py3-none-any.whl.metadata (3.6 kB)
Collecting langgraph<1.0,>=1.0.2 (from langchain->-r requirements.txt (line 1))
  Using cached langgraph-1.0.4-py3-none-any.whl.metadata (7.8 kB)
Collecting pydantic<3.0.0,>=2.7.4 (from langchain->-r requirements.txt (line 1))
  Using cached pydantic-2.12.5-py3-none-any.whl.metadata (90 kB)
Collecting jsonpatch<2.0.0,>=1.33.0 (from langchain-core<2.0.0,>=1.1.0->langchain->-r requirements.txt (line 1))
  Using cached jsonpatch-1.33-py2.py3-none-any.whl.metadata (3.0 kB)
Collecting langsmith<1.0.0,>=0.3.45 (from langchain-core<2.0.0,>=1.1.0->langchain->-r requirements.txt (line 1))
  Using cached langsmith-0.4.49-py3-none-any.whl.metadata (14 kB)
Collecting packaging<26.0.0,>=23.2.0 (from langchain-core<2.0.0,>=1.1.0->langchain->-r requirements.txt (line 1))
  Using cached packaging-25.0-py3-none-any.whl.metadata (3.3 kB)
Collecting pyyaml<7.0.0,>=5.3.0 (from langchain-core<2.0.0,>=1.1.0->langchain->-r requirements.txt (line 1))
  Using cached pyyaml-6.0.3-cp313-cp313-win_amd64.whl.metadata (2.4 kB)
```



The screenshot shows the Visual Studio Code editor with the same `requirements.txt` file. The terminal window at the bottom shows the command `$ ollama pull llama3` being executed. The output displays the progress of pulling the `llama3` model from Ollama, showing the manifest being pulled and the model files being downloaded in chunks.

```
requirements.txt
1 langchain
2 faiss-cpu
3 sentence-transformers
4 python-dotenv
5 ollama
6

(venv)
rbsru@sruthys_L0Q MINGW64 ~/OneDrive/Desktop/Project3-483
$ ollama pull llama3
pulling manifest
pulling 6a9746a1ec1a: 100% 4.7 GB
pulling 4fa551d4f938: 100% 12 KB
pulling 8ab4849b038c: 100% 254 B
pulling 577073ffcc6c: 100% 110 B
pulling 3f8ebda8a7fa: 100% 485 B
verifying sha256 digest
writing manifest
success

rbsru@sruthys_L0Q MINGW64 ~/OneDrive/Desktop/Project3-483
$
```

## 2.4 Document Loading and Preprocessing

The system loads a folder of .txt files that cover topics like neural networks, transformers, and AI in healthcare. Each file is read and split into overlapping chunks using LangChain’s `RecursiveCharacterTextSplitter`.

Configuration used:

Chunk size: 800 characters

Chunk overlap: 100

This keeps semantic continuity while still producing manageable embeddings.

## **2.5 Embeddings and FAISS Index**

Each chunk is embedded using sentence-transformers/all-MiniLM-L6-v2, a fast and lightweight embedding model. These embeddings are stored in a FAISS index, which allows for quick similarity searches.

If the FAISS index already exists, the program loads it; if not, it rebuilds the index. This makes the pipeline efficient for repeated use.

## **2.6 Retrieval and RAG Chain**

A retriever is created from the FAISS index using `as_retriever(search_kwargs={"k": 5})`.

When a query is made:

1. The user's question goes into the RAG chain.
2. The retriever selects the top five most relevant chunks.
3. A custom prompt is built dynamically:

“You are a highly intelligent research assistant.  
Use ONLY the retrieved context to answer the question.”

4. The prompt is sent to LLaMA 3 running locally via ChatOllama.
5. The answer and sources are returned.

This pipeline ensures that responses are based solely on the loaded documents.

## **2.7 Demonstration Script (run\_research\_assistant.py)**

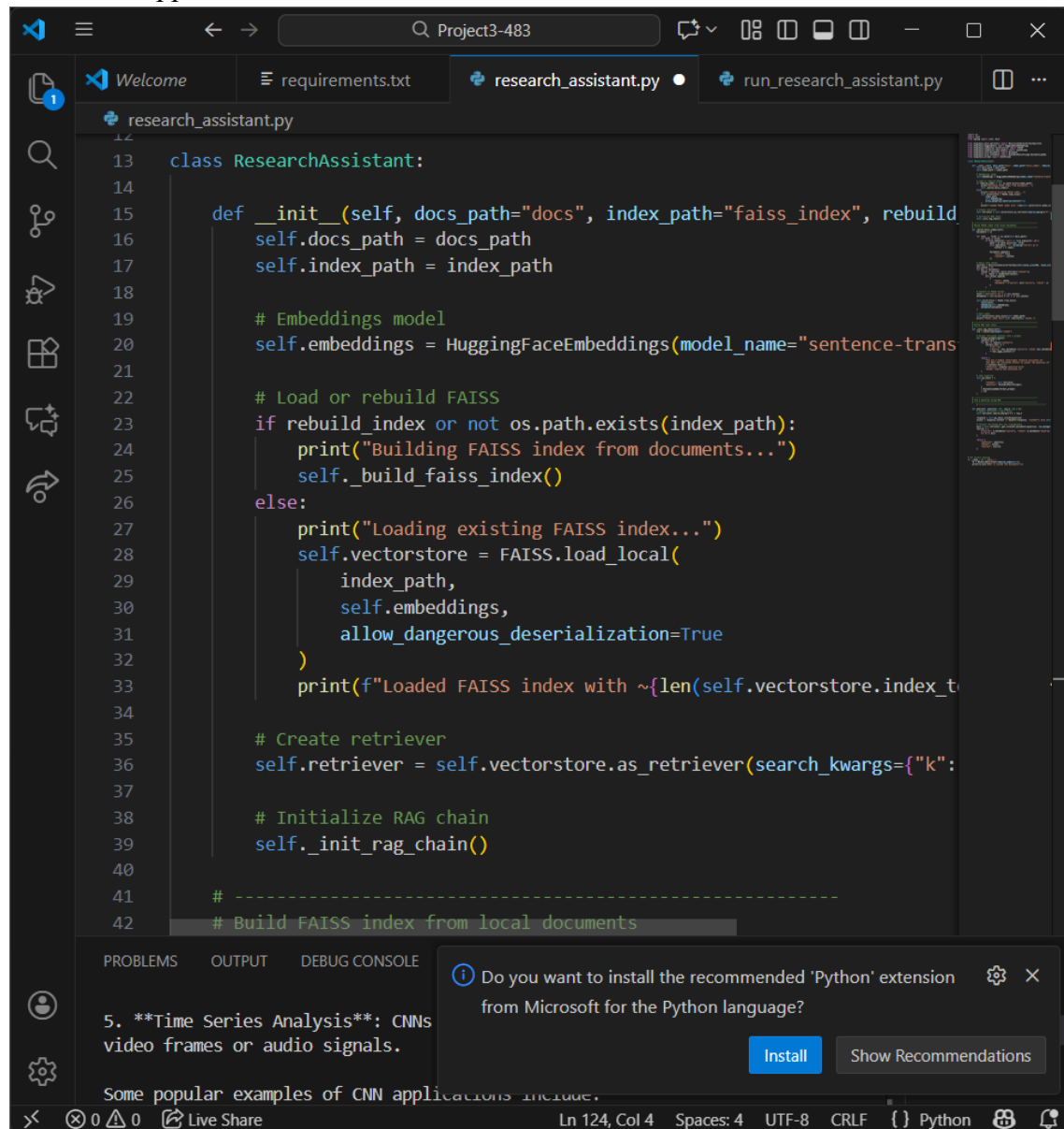
The test script shows three required interactions:

1. A general factual question
2. A document-based follow-up question
3. A summarization request

The script prints the question, answer, and retrieved source chunks, and saves the outputs to a JSON file.

### 3. Screenshots

#### A. Code Snippets



The screenshot shows a Visual Studio Code editor window with the file `research_assistant.py` open. The code defines a `ResearchAssistant` class with an `__init__` method. The method initializes the `docs_path` and `index_path` attributes, sets up a `HuggingFaceEmbeddings` model, and either builds or loads a FAISS index. It also creates a retriever and initializes a RAG chain. A comment at the bottom indicates the next step is to build the FAISS index from local documents.

```
13 class ResearchAssistant:
14
15     def __init__(self, docs_path="docs", index_path="faiss_index", rebuild_index=True):
16         self.docs_path = docs_path
17         self.index_path = index_path
18
19         # Embeddings model
20         self.embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
21
22         # Load or rebuild FAISS
23         if rebuild_index or not os.path.exists(index_path):
24             print("Building FAISS index from documents...")
25             self._build_faiss_index()
26         else:
27             print("Loading existing FAISS index...")
28             self.vectorstore = FAISS.load_local(
29                 index_path,
30                 self.embeddings,
31                 allow_dangerous_deserialization=True
32             )
33             print(f"Loaded FAISS index with ~{len(self.vectorstore.index_to_docstore_id)} documents")
34
35         # Create retriever
36         self.retriever = self.vectorstore.as_retriever(search_kwargs={"k": 5})
37
38         # Initialize RAG chain
39         self._init_rag_chain()
40
41         # -----
42         # Build FAISS index from local documents
```

At the bottom of the editor, a notification asks: "Do you want to install the recommended 'Python' extension from Microsoft for the Python language?". Below this, a snippet of text is visible: "5. \*\*Time Series Analysis\*\*: CNNs video frames or audio signals. Some popular examples of CNN applications include."

Initialization code (`__init__`)

```
13 class ResearchAssistant:
44     def _build_faiss_index(self):
45         documents = []
46
47         for root, _, files in os.walk(self.docs_path):
48             for file in files:
49                 if file.endswith(".txt") or file.endswith(".md"):
50                     path = os.path.join(root, file)
51                     with open(path, "r", encoding="utf-8") as f:
52                         content = f.read()
53
54                     documents.append({
55                         "source": file,
56                         "content": content
57                     })
58
59         # Split into chunks
60         splitter = RecursiveCharacterTextSplitter(chunk_size=800, chunk_ov
61         all_chunks = []
62         for doc in documents:
63             chunks = splitter.split_text(doc["content"])
64             for i, chunk in enumerate(chunks):
65                 all_chunks.append(
66                     {
67                         "text": chunk,
68                         "metadata": {"source": doc["source"], "chunk": i}
69                     }
70             )
71
72         # Convert to FAISS format
```

5. **Time Series Analysis**: CNNs process video frames or audio signals.

Some popular examples of CNN applications include:

Do you want to install the recommended 'Python' extension from Microsoft for the Python language?

[Install](#) [Show Recommendations](#)

Chunking / FAISS building code

```
13 class ResearchAssistant:
88     #
89     def _init_rag_chain(self):
90         llm = ChatOllama(model="llama3")
91
92         # Format retrieved context into a prompt
93     def format_prompt(inputs):
94         context_text = ""
95         for doc in inputs["context"]:
96             context_text += (
97                 f"\n---\n"
98                 f"Source: {doc.metadata['source']} (chunk {doc.metadata['chunk_id']})\n"
99                 f"{doc.page_content}\n"
100             )
101
102         return (
103             "You are a highly intelligent research assistant.\n"
104             "Use ONLY the retrieved context to answer the question.\n"
105             f"{context_text}\n"
106             f"Question: {inputs['question']}\n"
107             "Answer clearly and concisely.\n"
108         )
109
110     # LCEL pipeline
111     self.qa_chain = (
112         {
113             "context": self.retriever,
114             "question": RunnablePassthrough()
115         }
116         | RunnableLambda(format_prompt)
117         | llm
```

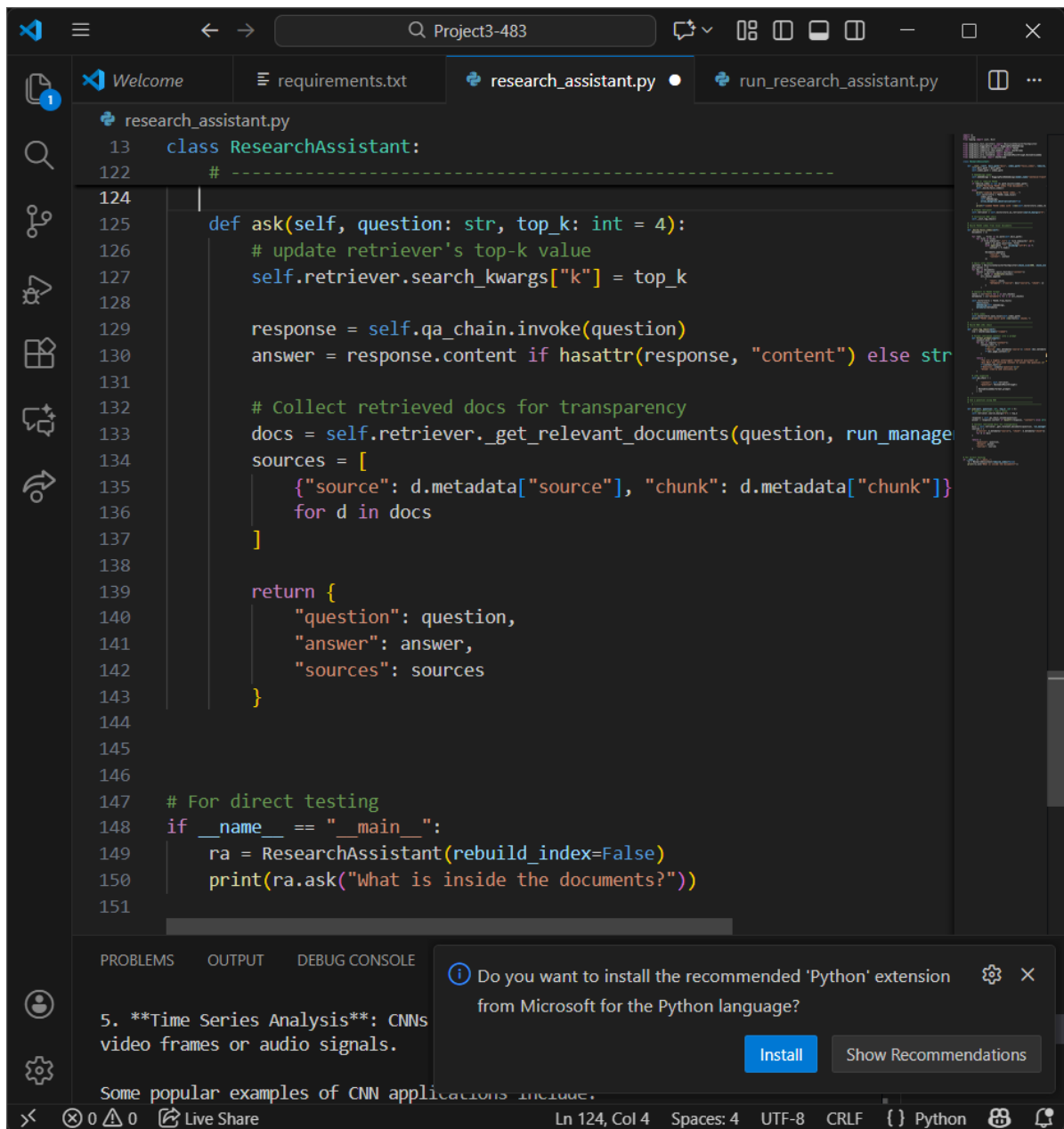
5. **\*\*Time Series Analysis\*\***: CNNs video frames or audio signals.

Some popular examples of CNN applications include.

Do you want to install the recommended 'Python' extension from Microsoft for the Python language?

Install Show Recommendations

RAG chain setup (`_init_rag_chain`)



```
13 class ResearchAssistant:
122 # -----
124
125 def ask(self, question: str, top_k: int = 4):
126     # update retriever's top-k value
127     self.retriever.search_kwargs["k"] = top_k
128
129     response = self.qa_chain.invoke(question)
130     answer = response.content if hasattr(response, "content") else str
131
132     # Collect retrieved docs for transparency
133     docs = self.retriever._get_relevant_documents(question, run_manage
134     sources = [
135         {"source": d.metadata["source"], "chunk": d.metadata["chunk"]}
136         for d in docs
137     ]
138
139     return {
140         "question": question,
141         "answer": answer,
142         "sources": sources
143     }
144
145
146
147 # For direct testing
148 if __name__ == "__main__":
149     ra = ResearchAssistant(rebuild_index=False)
150     print(ra.ask("What is inside the documents?"))
151
```

5. **Time Series Analysis**: CNNs  
video frames or audio signals.

Some popular examples of CNN applications include.

Do you want to install the recommended 'Python' extension from Microsoft for the Python language?

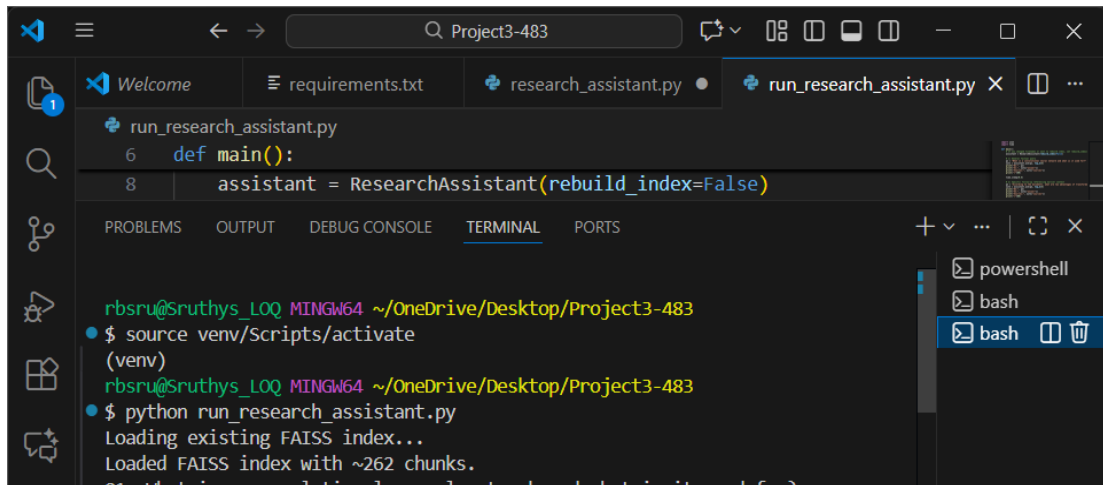
Install Show Recommendations

Ln 124, Col 4 Spaces: 4 UTF-8 CRLF Python

The ask() method



## B. Terminal Output



```
run_research_assistant.py
6 def main():
8     assistant = ResearchAssistant(rebuild_index=False)

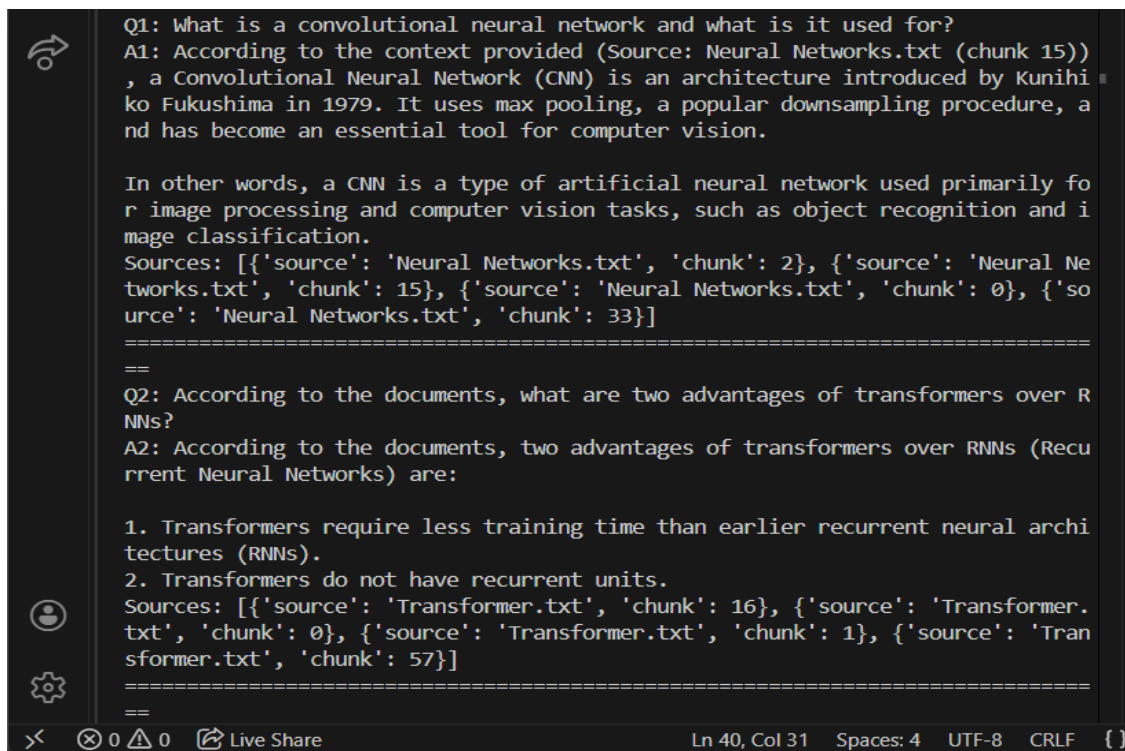
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

rbsru@Sruthys_LOQ MINGW64 ~/OneDrive/Desktop/Project3-483
$ source venv/Scripts/activate
(venv)
rbsru@Sruthys_LOQ MINGW64 ~/OneDrive/Desktop/Project3-483
$ python run_research_assistant.py
Loading existing FAISS index...
Loaded FAISS index with ~262 chunks.
```

FAISS loading message:

Loading existing FAISS index...

Loaded FAISS index with ~262 chunks.



```
Q1: What is a convolutional neural network and what is it used for?
A1: According to the context provided (Source: Neural Networks.txt (chunk 15))
, a Convolutional Neural Network (CNN) is an architecture introduced by Kunihi
ko Fukushima in 1979. It uses max pooling, a popular downsampling procedure, a
nd has become an essential tool for computer vision.

In other words, a CNN is a type of artificial neural network used primarily fo
r image processing and computer vision tasks, such as object recognition and i
mage classification.
Sources: [{'source': 'Neural Networks.txt', 'chunk': 2}, {'source': 'Neural Ne
tworks.txt', 'chunk': 15}, {'source': 'Neural Networks.txt', 'chunk': 0}, {'so
urce': 'Neural Networks.txt', 'chunk': 33}]

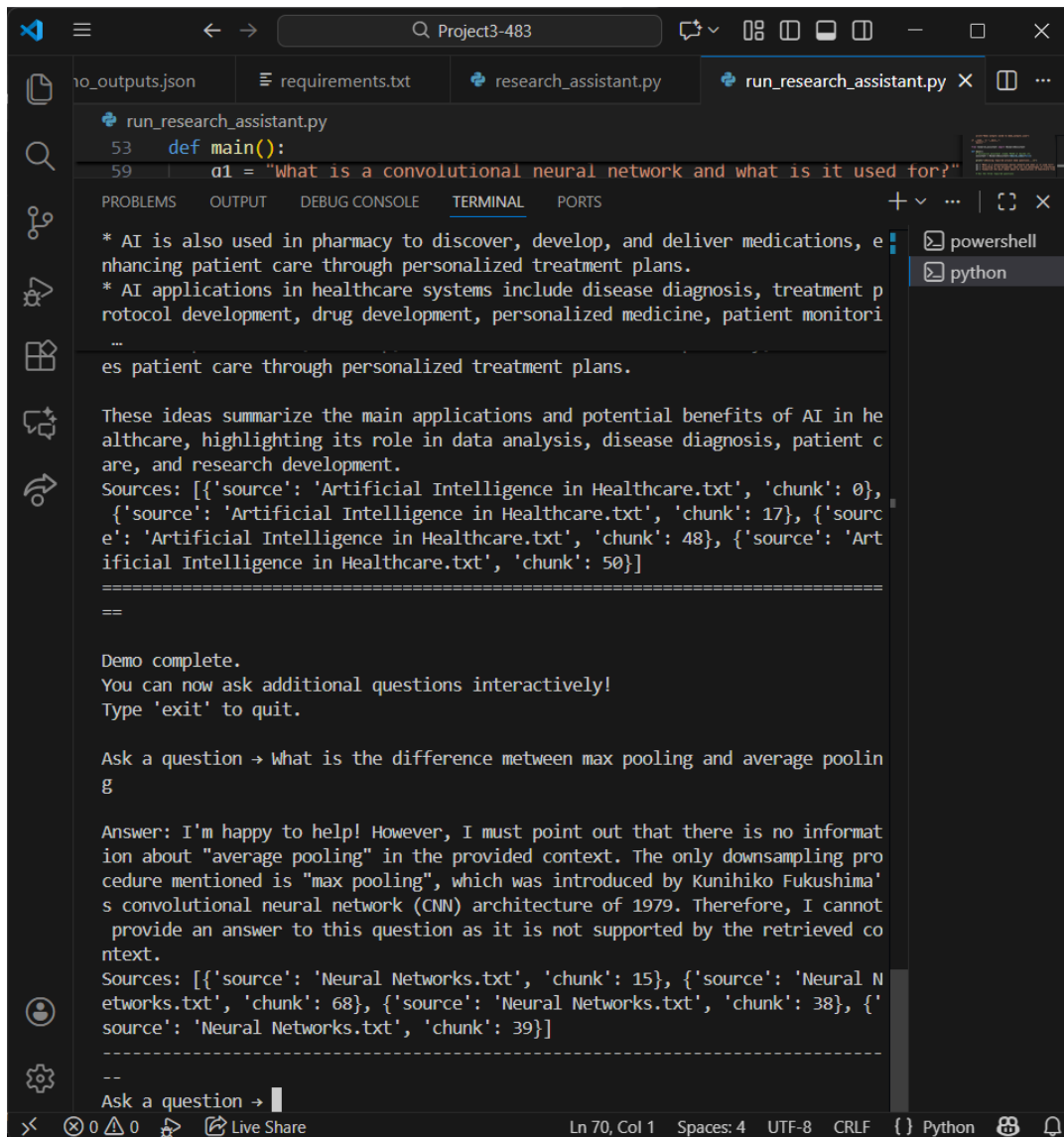
==

Q2: According to the documents, what are two advantages of transformers over R
NNs?
A2: According to the documents, two advantages of transformers over RNNs (Recu
rrent Neural Networks) are:

1. Transformers require less training time than earlier recurrent neural archi
tectures (RNNs).
2. Transformers do not have recurrent units.
Sources: [{'source': 'Transformer.txt', 'chunk': 16}, {'source': 'Transformer.
txt', 'chunk': 0}, {'source': 'Transformer.txt', 'chunk': 1}, {'source': 'Tran
sformer.txt', 'chunk': 57}]

==
```





The image shows a Visual Studio Code editor window with a project named "Project3-483". The editor has several tabs open: "io\_outputs.json", "requirements.txt", "research\_assistant.py", and "run\_research\_assistant.py". The "run\_research\_assistant.py" tab is active, showing a Python script with a `main()` function. The script's output is displayed in the terminal pane at the bottom, which is currently set to "python" mode. The output shows a list of sources, a demo completion message, and an answer to a question about max pooling and average pooling.

```
run_research_assistant.py
53 def main():
59     q1 = "What is a convolutional neural network and what is it used for?"

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
* AI is also used in pharmacy to discover, develop, and deliver medications, e
nhancing patient care through personalized treatment plans.
* AI applications in healthcare systems include disease diagnosis, treatment p
rotocol development, drug development, personalized medicine, patient monitori
...
es patient care through personalized treatment plans.

These ideas summarize the main applications and potential benefits of AI in he
althcare, highlighting its role in data analysis, disease diagnosis, patient c
are, and research development.
Sources: [{'source': 'Artificial Intelligence in Healthcare.txt', 'chunk': 0},
{'source': 'Artificial Intelligence in Healthcare.txt', 'chunk': 17}, {'sourc
e': 'Artificial Intelligence in Healthcare.txt', 'chunk': 48}, {'source': 'Art
ificial Intelligence in Healthcare.txt', 'chunk': 50}]
=====
==

Demo complete.
You can now ask additional questions interactively!
Type 'exit' to quit.

Ask a question -> What is the difference between max pooling and average poolin
g

Answer: I'm happy to help! However, I must point out that there is no informat
ion about "average pooling" in the provided context. The only downsampling pro
cedure mentioned is "max pooling", which was introduced by Kunihiko Fukushima'
s convolutional neural network (CNN) architecture of 1979. Therefore, I cannot
provide an answer to this question as it is not supported by the retrieved co
ntext.
Sources: [{'source': 'Neural Networks.txt', 'chunk': 15}, {'source': 'Neural N
etworks.txt', 'chunk': 68}, {'source': 'Neural Networks.txt', 'chunk': 38}, {'
source': 'Neural Networks.txt', 'chunk': 39}]
-----
--
Ask a question ->
```

Output for Q1, Q2, and Q3

#### 4. Demonstration Results (Actual Outputs)

```
{
  "interaction_1": {
    "question": "What is a convolutional neural network and what is it used for?",
    "answer": "According to the context provided (Source: Neural Networks.txt (chunk 15)), a Convolutional Neural Network (CNN) is an architecture introduced by Kunihiro Fukushima in 1979. It uses max pooling, a popular downsampling procedure, and has become an essential tool for computer vision.\n\nIn other words, a CNN is a type of artificial neural network used primarily for image processing and computer vision tasks, such as object recognition and image classification.",
    "sources": [
      {
        "source": "Neural Networks.txt",
        "chunk": 2
      },
      {
        "source": "Neural Networks.txt",
        "chunk": 15
      },
      {
        "source": "Neural Networks.txt",
        "chunk": 0
      },
      {
        "source": "Neural Networks.txt",
        "chunk": 33
      }
    ]
  },
  "interaction_2": {
    "question": "According to the documents, what are two advantages of transformers over RNNs?",
    "answer": "According to the documents, two advantages of transformers over RNNs (Recurrent Neural Networks) are:\n\n1. Transformers require less training time than earlier recurrent neural architectures (RNNs).\n2. Transformers do not have recurrent units.",
    "sources": [
      {
        "source": "Transformer.txt",
```

```

    "chunk": 16
  },
  {
    "source": "Transformer.txt",
    "chunk": 0
  },
  {
    "source": "Transformer.txt",
    "chunk": 1
  },
  {
    "source": "Transformer.txt",
    "chunk": 57
  }
]
},
"interaction_3": {
  "question": "Summarize the main ideas about AI applications in healthcare from the loaded files.",
  "answer": "Based on the retrieved context, the main ideas about AI applications in healthcare are:\n\n* AI can analyze complex medical data to diagnose, treat, or prevent diseases faster and more accurately than humans.\n* AI can automate administrative tasks, prioritize patient needs, and facilitate seamless communication among healthcare teams.\n* AI can be used in breast imaging for analyzing screening mammograms and improving breast cancer detection rates.\n* AI can help discover, develop, and deliver medications, as well as enhance patient care through personalized treatment plans.\n* AI has the potential to streamline care coordination, reduce workload, and improve patient monitoring and care.\n* AI can assist clinicians with disease diagnosis by processing large amounts of electronic health records (EHRs) and analyzing symptoms.\n* AI can also aid in early prediction of diseases such as Alzheimer's and dementia.\n\nThese applications have the potential to revolutionize healthcare systems, especially in developing nations where resources may be scarce.",
  "sources": [
    {
      "source": "Artificial Intelligence in Healthcare.txt",
      "chunk": 0
    },
    {
      "source": "Artificial Intelligence in Healthcare.txt",
      "chunk": 17
    }
  ]
}

```

```

    },
    {
      "source": "Artificial Intelligence in Healthcare.txt",
      "chunk": 48
    },
    {
      "source": "Artificial Intelligence in Healthcare.txt",
      "chunk": 50
    },
    {
      "source": "Artificial Intelligence in Healthcare.txt",
      "chunk": 3
    },
    {
      "source": "Artificial Intelligence in Healthcare.txt",
      "chunk": 55
    }
  ]
}

```

## 5. Reflection

Transformers and RNNs are two different approaches to sequence modeling, and this project shows why modern systems rely heavily on Transformer architectures along with retrieval methods.

RNNs, such as LSTMs and GRUs, process information sequentially, maintaining a hidden state from one step to the next. This creates a bottleneck: early information must be compressed into a single state vector. As sequences get longer, the model finds it hard to keep information because gradients fade over time. Even structures designed to address this, like LSTMs, still face long-range dependency issues.

Transformers replace recurrence with self-attention, allowing every input token to connect directly to every other token. Instead of remembering details step-by-step, the model calculates relationships in parallel, making it more efficient and significantly better at capturing long-range structures. However, Transformers have a fixed context window and cannot natively handle documents longer than a certain number of tokens.

This is where Retrieval-Augmented Generation (RAG) becomes important. By embedding documents and storing them in FAISS, the system retrieves only the most relevant text when a question is asked. The model then receives the retrieved chunks directly in the prompt, effectively giving it access to information far beyond its context limits.

This hybrid system combines:

- the flexible, non-sequential reasoning of Transformers,
- with the long-term memory provided by vector retrieval.

In practice, RAG greatly extends an LLM's usable memory while keeping inference efficient. This setup shows why modern AI systems rely less on internal sequence memory (like RNNs) and more on structured external memory systems that allow for more accurate and context-rich responses.

## **6. Conclusion**

This project successfully created a local RAG-based research assistant using LangChain and Ollama. By integrating text splitting, embeddings, FAISS retrieval, and a custom prompt pipeline, the system showed how retrieval can enhance a Transformer model's reasoning abilities.

Through three test interactions—factual queries, context-aware follow-ups, and summarization—the system demonstrated correct retrieval behavior and grounded answers based on the loaded documents. The project provided practical experience with LLM pipelines, vector databases, and modern retrieval methods commonly used in real-world AI applications.