

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

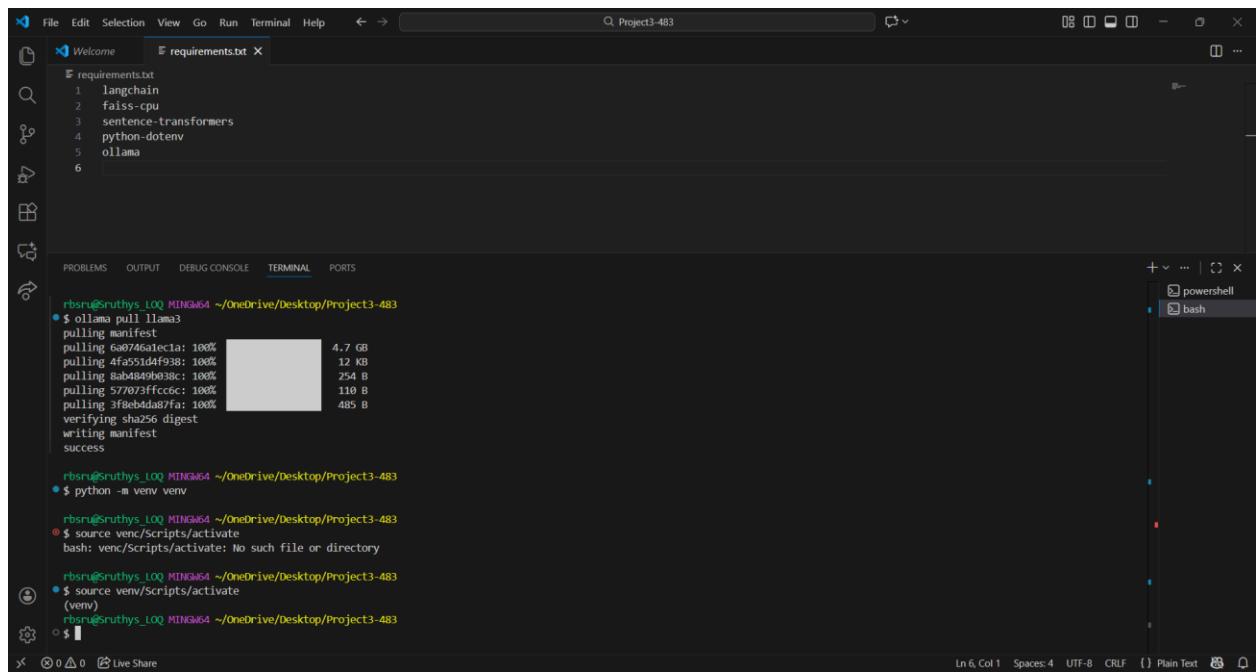
ollama pull llama3

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



The screenshot shows the VS Code interface with the terminal tab active. The terminal window displays the following command and its execution:

```
rbsru@Sruthys-LOO MINGW64 ~/OneDrive/Desktop/Project3-483
$ ollama pull llama3
pulling manifest
pulling 6a0746a1ec1a: 100%
pulling 4fa551d4f938: 100%
pulling 8ab4b49fb038c: 100%
pulling 577073ffccbc: 100%
pulling 3f8e8dd4da87fa: 100%
verifying sha256 digest
writing manifest
success

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

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

```

File Edit Selection View Go Run Terminal Help ← → Project3-483
Welcome requirements.txt
requirements.txt
1 langchain
2 faiss-cpu
3 sentence-transformers
4 python-dotenv
5 ollama
6

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(venv)
rbsru@Sruthys-LOQ 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 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 langgraphv1.1.0,>=1.0.2 (from langchain->r requirements.txt (line 1))
  Using cached langgraphv1.1.0-py3-none-any.whl.metadata (7.9 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 (99 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<0.0.0,>=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,>=25.2.0 (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)

```

```

File Edit Selection View Go Run Terminal Help ← → Project3-483
Welcome requirements.txt
requirements.txt
1 langchain
2 faiss-cpu
3 sentence-transformers
4 python-dotenv
5 ollama
6

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
rbsru@Sruthys-LOQ MINGW64 ~/OneDrive/Desktop/Project3-483
$ ollama pull llama3
pulling manifest
pulling 6a07461e1c1a: 100%
pulling 4f4551d4f938: 100%
pulling 8ab4b49b038c: 100%
pulling 577073ffcc6c: 100%
pulling 3f8eb4da87fa: 100%
verifying sha256 digest
writing manifest
success

rbsru@Sruthys-LOQ 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`)

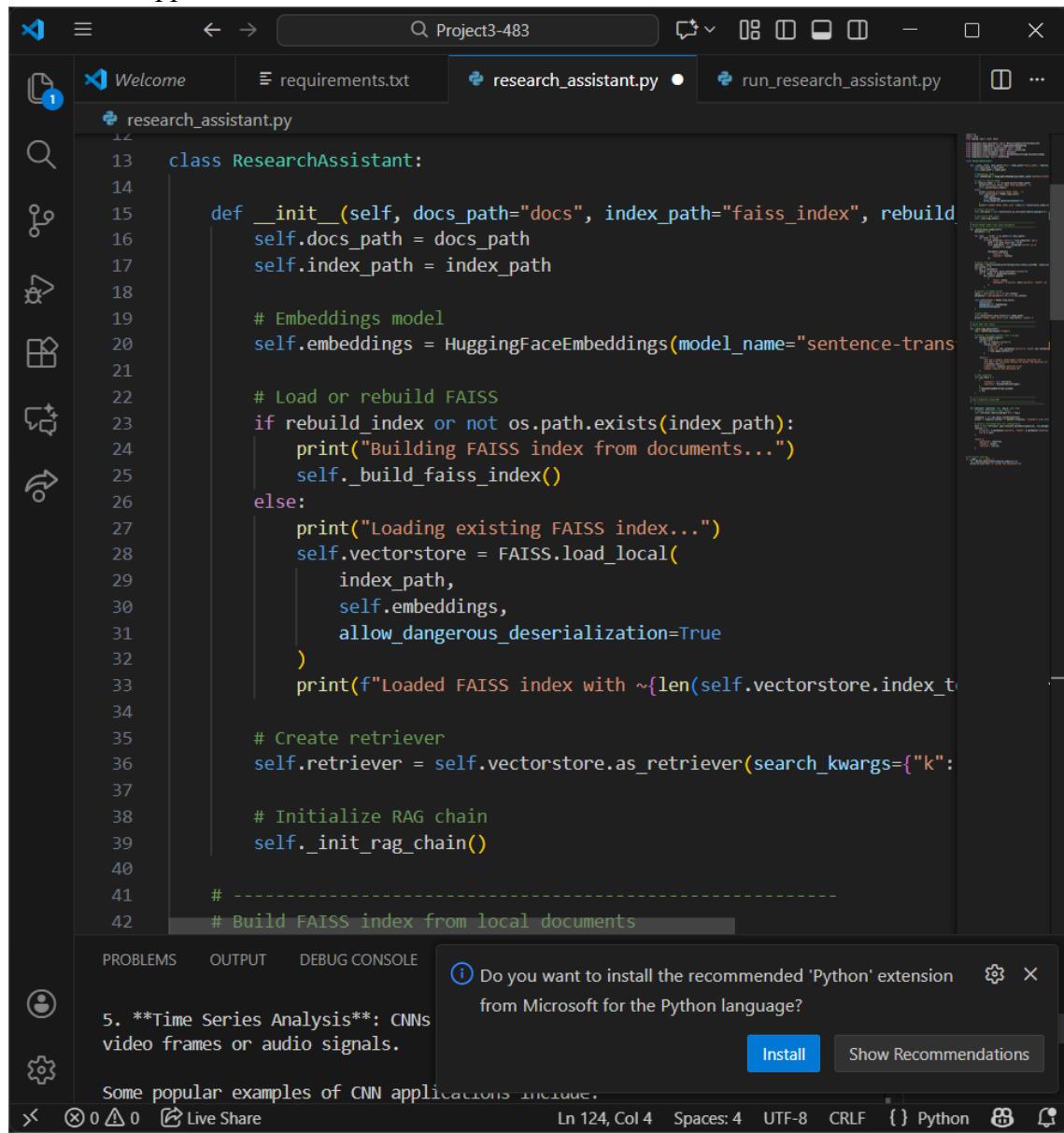
The test script shows three required interactions:

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 dark-themed instance of Visual Studio Code (VS Code) with the title bar "Project3-483". The left sidebar contains icons for file operations like Open, Save, Find, and others. The main editor area displays Python code for a "research_assistant.py" file. The code defines a "ResearchAssistant" class with an "__init__" method. This method initializes attributes for document and index paths, sets up an embeddings model using HuggingFaceEmbeddings, and either builds or loads an FAISS index. It also creates a retriever and initializes a RAG chain. A tooltip at the bottom right of the editor asks if the user wants to install the recommended 'Python' extension from Microsoft for the Python language, with "Install" and "Show Recommendations" buttons. Below the editor, the status bar shows "Ln 124, Col 4" and other file-related information. The bottom navigation bar includes "PROBLEMS", "OUTPUT", and "DEBUG CONSOLE" tabs, along with "Live Share" and other status indicators.

```
13     class ResearchAssistant:
14
15         def __init__(self, docs_path="docs", index_path="faiss_index", rebuild=False):
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 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)} documents")
34
35             # Create retriever
36             self.retriever = self.vectorstore.as_retriever(search_kwargs={"k": 3})
37
38             # Initialize RAG chain
39             self._init_rag_chain()
40
41             # -----
42             # Build FAISS index from local documents
```

Initialization code (`__init__`)

The screenshot shows the Microsoft Visual Studio Code interface with the following details:

- Title Bar:** Project3-483
- File Explorer:** Shows files: Welcome, requirements.txt, research_assistant.py (selected), run_research_assistant.py.
- Code Editor:** Displays Python code for a `ResearchAssistant` class. The code reads documents from a directory, splits them into chunks, and converts them to FAISS format. A tooltip at the bottom of the editor says "# Convert to FAISS format".
- Bottom Status Bar:** PROBLEMS, OUTPUT, DEBUG CONSOLE, 5. **Time Series Analysis**: CNNs video frames or audio signals., Some popular examples of CNN applications include., Ln 124, Col 4, Spaces: 4, UTF-8, CRLF, {}, Python, Live Share.
- Bottom Right Pop-up:** A recommendation to install the 'Python' extension from Microsoft for the Python language, with "Install" and "Show Recommendations" buttons.

Chunking / FAISS building code

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** Project3-483
- Left Sidebar:** Includes icons for File, Find, Replace, Go To, Open, Save, and others.
- Top Bar:** Shows tabs for "Welcome", "requirements.txt", "research_assistant.py", and "run_research_assistant.py".
- Code Editor:** Displays Python code for a "ResearchAssistant" class. The code includes methods for initializing an LLM, formatting prompts, and defining a pipeline for question answering.
- Bottom Status Bar:** Shows "PROBLEMS", "OUTPUT", "DEBUG CONSOLE", and a message about Time Series Analysis.
- Bottom Right:** A modal dialog box asks if the user wants to install the recommended 'Python' extension from Microsoft for the Python language. It has "Install" and "Show Recommendations" buttons.

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_manager)
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
```

PROBLEMS OUTPUT DEBUG CONSOLE

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

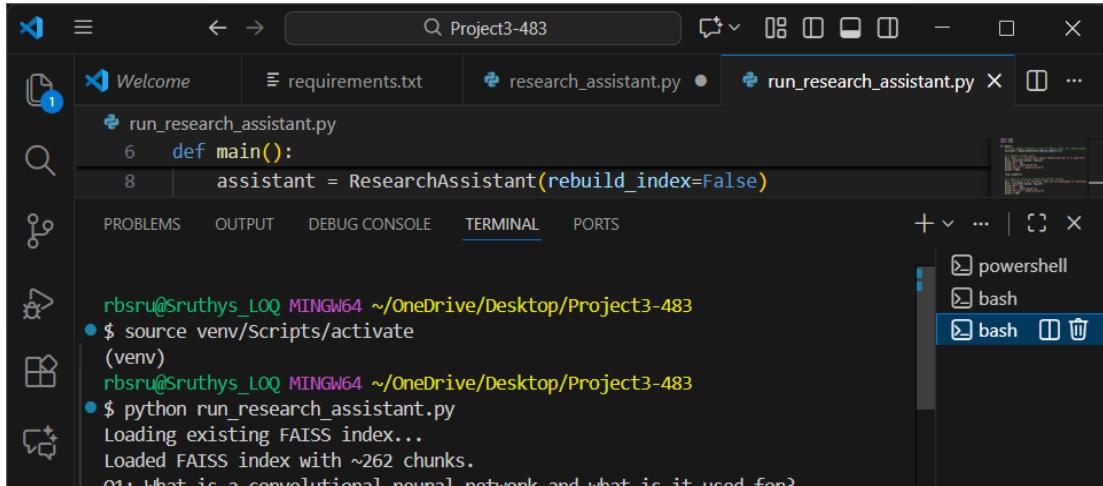
Some popular examples of CNN applications include:

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

Install Show Recommendations

The ask() method

B. Terminal Output



The screenshot shows a terminal window in VS Code with the title bar "Project3-483". The terminal tab is selected. The code editor shows a file named "run_research_assistant.py" with the following content:

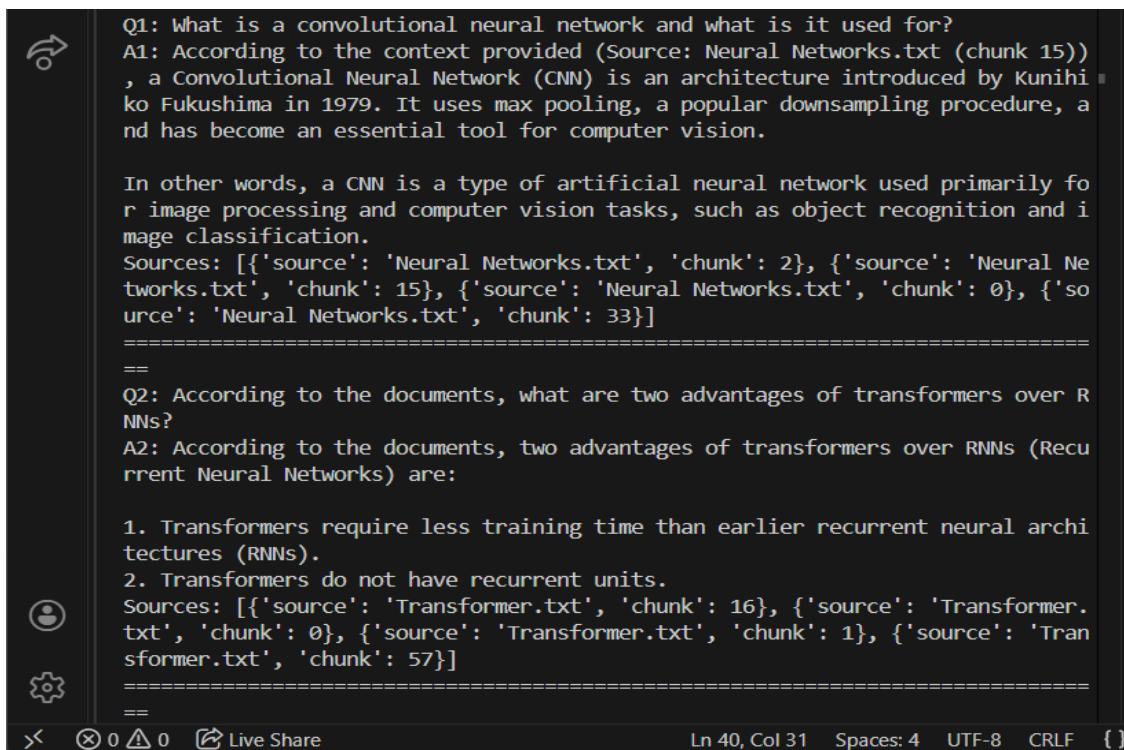
```
def main():
    assistant = ResearchAssistant(rebuild_index=False)
```

The terminal output shows the following command-line session:

```
rbsru@Sruthys-LQ: MINGW64 ~/OneDrive/Desktop/Project3-483
$ source venv/Scripts/activate
(venv)
rbsru@Sruthys-LQ: 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.



The screenshot shows a terminal window in VS Code with the title bar "Project3-483". The terminal tab is selected. The code editor shows a file named "run_research_assistant.py" with the following content:

```
def main():
    assistant = ResearchAssistant(rebuild_index=False)
```

The terminal output shows a conversation with a research assistant:

```
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 Kunihiko Fukushima in 1979. It uses max pooling, a popular downsampling procedure, and has become an essential tool for computer vision.

In 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}]
=====
==

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

1. Transformers require less training time than earlier recurrent neural architectures (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": "Transformer.txt", "chunk": 57}]
=====
```

The screenshot shows a terminal window within a dark-themed IDE interface. The title bar reads "Project3-483". The terminal tab is active, displaying Python code for generating AI responses. The code defines a function `main()` that prints a question and then generates AI answers based on the question. The first answer is a continuation of the question, while subsequent answers provide general information about AI applications in healthcare. A context menu is open over the AI-generated text, showing options like "Navigate to Command", "Scroll to Previous Command (Ctrl+UpArrow)", and "Scroll to Next Command (Ctrl+DownArrow)". The bottom status bar shows file paths, line and column numbers (Ln 70, Col 1), and encoding information (UTF-8, CRLF). The bottom right corner includes icons for Python, GitHub, and a profile picture.

```
run_research_assistant.py
53 def main():
54     q1 = "What is a convolutional neural network and what is it used for?"
55
56     print(q1)
57
58     * AI is also used in pharmacy to discover, develop, and deliver medications, e
59     nhancing patient care through personalized treatment plans
60     * AI applications in healthcare systems include disease di
61     rotocol development, drug development, personalized medic
62     ...
63
64     Q3: Summarize the main ideas about AI applications in healthcare from the load
65     ed files.
66     A3: Based on the retrieved context, the main ideas about AI applications in he
67     althcare are:
68
69     1. AI can analyze and understand complex medical data, exceeding or augmenting
70        human capabilities to diagnose, treat, or prevent disease.
71     2. AI programs are being applied across various medical subdisciplines, includ
72        ing diagnostics, treatment protocol development, drug development, personaliz
73        e medicine, and patient monitoring and care.
74     3. AI has the potential to streamline care coordination, reduce workload, and
75        automate administrative tasks in healthcare.
76     4. AI is used in breast imaging for analyzing screening mammograms, improving
77        breast cancer detection rates, and reducing radiologist's reading workload.
78     5. The FDA-approved AI-enabled medical devices are primarily in radiology (77%
79        as of 2025).
80     6. AI helps discover, develop, and deliver medications in pharmacy, and enhanc
81        es patient care through personalized treatment plans.
82
83     These ideas summarize the main applications and potential benefits of AI in he
84     althcare, highlighting its role in data analysis, disease diagnosis, patient c
85     are, and research development.
86     Sources: [{"source": "Artificial Intelligence in Healthcare.txt", "chunk": 0},
87     {"source": "Artificial Intelligence in Healthcare.txt", "chunk": 17}, {"sourc
88     e": "Artificial Intelligence in Healthcare.txt", "chunk": 48}, {"source": "Art
89     ificial Intelligence in Healthcare.txt", "chunk": 50}]
90 =====
91 ==
92
93 Demo complete.
```

The screenshot shows a terminal window in Visual Studio Code (VS Code) with the title bar "Project3-483". The terminal tab is active, displaying Python code and AI-generated responses.

```
run_research_assistant.py
53 def main():
59     a1 = "What is a convolutional neural network and what is it used for?"
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ... | X
* AI is also used in pharmacy to discover, develop, and deliver medications, enhancing patient care through personalized treatment plans.
* AI applications in healthcare systems include disease diagnosis, treatment protocol development, drug development, personalized medicine, patient monitoring, and research development.
These ideas summarize the main applications and potential benefits of AI in healthcare, highlighting its role in data analysis, disease diagnosis, patient care, and research development.
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}]
=====
==

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 pooling
Answer: I'm happy to help! However, I must point out that there is no information about "average pooling" in the provided context. The only downsampling procedure 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 context.
Sources: [{"source": "Neural Networks.txt", "chunk": 15}, {"source": "Neural Networks.txt", "chunk": 68}, {"source": "Neural Networks.txt", "chunk": 38}, {"source": "Neural Networks.txt", "chunk": 39}]
=====
```

The terminal shows a Python script running, displaying AI-generated responses to questions about AI in healthcare and the difference between max and average pooling. The interface includes standard VS Code icons for file operations, search, and terminal navigation.

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 Kunihiko  
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:\n1. Transformers require less training time than  
earlier recurrent neural architectures (RNNs).\n2. Transformers do not have recurrent  
units.",  
    "sources": [  
      {  
        "source": "Transformer.txt",  
        "chunk": 0  
      }  
    ]  
  }  
}
```

```
        "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.