Basic RAG Implementation and Comparison

Chong Chen

October 25, 2024

Abstract

In this assignment, I developed a basic Retrieval-Augmented Generation (RAG) system and compared its performance with standalone local language models, using the existing interface. All code and test results are available on GitHub, and a video demonstration can be viewed on YouTube.

1 Implementation and Integration of RAG System

1.1 System Setup and Configuration

The Retrieval-Augmented Generation (RAG) system was implemented using the LlamaIndex and Ollama libraries, both configured to run on a local machine. The system integrates a locally hosted large language model such as Llama3 or Gemma2 and an embedding model (`nomic-embed-text`). It leverages a VectorStoreIndex from the LlamaIndex library to process and store user-uploaded documents as vectors, enabling relevant information retrieval during conversations.

A key feature of this system is its ability to switch dynamically between two operational modes:

- **RAG Mode:** Augments chatbot responses by retrieving and integrating context from uploaded documents.
- **Non-RAG Mode:** Generates responses solely based on the capabilities of the local LLM, without document retrieval.

1.2 Local System Functionality

The RAG system is designed to handle a variety of file types, including `.txt`, `.pdf`, and `.docx`. The system ensures smooth operation by managing parallel processes and limiting the number of concurrently loaded models, preventing the overutilization of local resources.

The system was tested locally using Streamlit as the front-end interface. Through the `setup_sidebar` function in `ui.py`, users can upload documents and toggle between RAG and non-RAG modes. Uploaded documents are processed using the `SimpleDirectoryReader` and stored as a `VectorStoreIndex`, allowing the LLM to retrieve relevant information from them during conversations.

1.3 User Interface Integration

The RAG system is fully integrated into the existing **Streamlit** interface, providing an intuitive user experience. Users interact with the chatbot via a toggle button in the sidebar, allowing them to switch between RAG and non-RAG modes easily. Additional interface features include:

- File uploader supporting `.txt`, `.pdf`, and `.docx` formats.
- Adjustable parameters for `max_length` and `temperature` via sidebar sliders.
- Buttons for starting new chats or clearing chat history.

Robust error handling ensures smooth operation, addressing issues such as missing file uploads or invalid model configurations.

1.4 Code Organization and Structure

The system follows a modular design pattern to ensure maintainability and extensibility. Each major functionality is encapsulated in dedicated modules:

- `config.py`: Centralized configuration settings.
- `utils.py`: Handles utility functions such as file management.
- `models.py`: Contains model initialization logic for both RAG and non-RAG modes.
- `ui.py`: Manages the user interface and chat history.
- `rag_module.py` and `non_rag_module.py`: Implement the logic for RAG and non-RAG modes, respectively.
- `app.py`: Serves as the entry point for launching the application.

The modular design improves code readability, facilitates debugging, and makes the system highly extendable. Exception handling is implemented throughout to handle potential issues such as missing LLM installations or corrupted document uploads.

2 Comparative Analysis and Reflection

2.1 Quality and diversity of test queries

The set of questions is designed to assess a wide range of knowledge in both the historical and technical domains of computer science. The questions are divided into two distinct categories: historical developments in computing and general computer science concepts.

The first six questions are derived from the document titled *History of computer and its generations*, which covers significant milestones in the development of computers and related technologies. These questions focus on key events such as the development of early computing systems, networking breakthroughs, and technological advancements that shaped the evolution of computers. For example, Question 1 evaluates the knowledge of the period when the UNIVAC 1107 was developed, while Question 2 explores the impact of Robert Metcalfe's development of Ethernet on computer interconnectivity. These questions help gauge an understanding of historical context and the progression of hardware and software technologies.

The remaining four questions are based on general computer science concepts, including data structures, distributed systems, and computer architecture. For instance, Question 7 addresses the properties of Red-Black trees, a self-balancing binary search tree used in computer science, while Question 8 focuses on the Paxos consensus algorithm, a key component in distributed systems. The variety in these topics ensures that the test covers both theoretical and practical aspects of computing.

Together, this set of queries not only evaluates a candidate's comprehension of historical developments in computing but also challenges their understanding of core computer science principles, offering a balanced and diverse assessment.

2.2 QA Result

In this section, I present the results of the QA performance evaluation for the four systems: Llama3 standalone, Llama3 with RAG, Gemma2 standalone, and Gemma2 with RAG. The accuracy percentages achieved by each system are as follows:

- **Llama3 (standalone)**: 70%
- **Llama3 + RAG**: 90%
- **Gemma2 (standalone)**: 60%
- **Gemma2 + RAG**: 85%

2.2.1 Llama3 Standalone vs. Llama3 + RAG

The standalone Llama3 system achieved an accuracy of 70%, demonstrating reasonable performance in responding to a range of factual and technical questions. However, when augmented with the RAG (Retrieval-Augmented Generation) approach, Llama3's performance saw a significant improvement, reaching an accuracy of 90%. The addition of RAG enabled the system to retrieve and integrate external knowledge, which helped in providing more accurate, relevant, and context-aware responses.

2.2.2 Gemma2 Standalone vs. Gemma2 + RAG

Gemma2, when used in its standalone form, achieved an accuracy of 60%, showing lower performance compared to Llama3. This suggests that Gemma2 has relatively more difficulty in handling complex or factually detailed questions. However, with the integration of RAG, Gemma2's performance jumped to 85%, demonstrating that retrieval augmentation had a substantial impact on improving the quality of its responses. Although it did not reach the performance level of Llama3 + RAG, it showed a marked improvement compared to its standalone version.

2.2.3 Overall Comparison

The results clearly indicate the effectiveness of RAG in both models. The Llama3 system consistently outperformed Gemma2, both in standalone and RAG-augmented modes. Llama3 + RAG achieved the highest score of 90%, while Gemma2 + RAG reached 85%, showing that RAG significantly enhanced both models' accuracy. However, the standalone versions of both models (Llama3 at 70% and Gemma2 at 60%) highlight that Llama3 is inherently better at handling QA tasks compared to Gemma2, even without external retrieval support.

2.2.4 Conclusion

The integration of RAG proves to be a highly effective method for improving QA performance, with Llama3 + RAG achieving the highest overall accuracy. Despite Gemma2 showing weaker standalone performance, the RAG augmentation brought it closer to Llama3's results, emphasizing the role of external knowledge retrieval in enhancing the accuracy and context-awareness of these systems.

2.3 Response Comparison(llama3 v.s. llama3 + RAG)

In this section, I compare the responses generated by the Llama3-only system and the Llama3 + RAG system based on the dimensions of relevance, accuracy, and context-awareness.

2.3.1 Relevance

The Llama3-only system struggles with relevance in certain instances. For example, when asked about the period during which the UNIVAC 1107 was developed, the system incorrectly associates it with the era of vacuum tubes (option a: 1937-1946). This shows that its response was not well-aligned with the context of the question. The Llama3 + RAG system, on the other hand, provides a more relevant response by identifying the correct period (1947-1962) and linking the development of the UNIVAC 1107 to the use of transistors, which is in line with historical records.

2.3.2 Accuracy

The Llama3-only system demonstrates several inaccuracies in its responses. For instance, it incorrectly places the development of the UNIVAC 1107 in the vacuum tube era and provides wrong answers for questions related to quantum computing advancements and the use of certain technologies. In contrast, the Llama3 + RAG system improves accuracy by leveraging external data. For example, when discussing the same UNIVAC 1107 question, it correctly places the computer in the era of transistors. Furthermore, in cases where the timeline or facts are unclear (e.g., quantum computing advancements in 2016), the RAG system attempts to clarify or correct itself based on available data, making it more reliable.

2.3.3 Context-Awareness

The Llama3-only system exhibits limited context-awareness, especially when handling questions that require deeper historical or technical understanding. For example, in the Red-Black tree question, the system provides a superficial answer without fully engaging with the specifics of the tree's properties. The Llama3 + RAG system, however, significantly improves context-awareness. It not only retrieves relevant documents but also interprets them in a way that enhances its ability to provide contextually appropriate responses. When handling technical questions, such as those related to distributed systems or historical developments, it adapts its explanations to the broader context, resulting in more coherent and insightful responses.

2.3.4 Summary

Overall, the Llama3 + RAG system outperforms the Llama3-only system in all dimensions. The retrieval-based augmentation allows the Llama3 + RAG system to provide more relevant, accurate, and context-aware responses by incorporating external knowledge into its answers. This improvement results in an overall accuracy gain from 70% (Llama3-only) to 90% (Llama3 + RAG).

2.4 Performance Analysis

I conducted a thorough analysis of response time and resource usage:

- **Response Time:** The average response time in RAG mode was approximately 2.5x to 5x slower compared to non-RAG mode, as the system had to retrieve and process external documents. Non-RAG mode responded in 3 to 20 seconds, while RAG mode took 30 to 50 seconds depending on the document size and complexity of the query.
- **CPU Usage:** RAG mode consumed approximately 10% more CPU resources during active document retrieval. Non-RAG mode maintained a lower CPU load.
- **Memory Usage:** The memory usage in RAG mode was significantly higher due to document indexing and LLM loading, typically ranging from 15-20% more than non-RAG mode.

2.5 Reflection on RAG System Capabilities

The RAG system offers significant improvements for queries that require external document context, such as answering questions based on uploaded files or summarizing long documents. However, the added latency and resource consumption should be noted. The system can be highly useful for users who need enhanced retrieval in specialized domains, but for general-purpose conversation, the non-RAG mode may be more efficient.

One of the key insights from the project is that RAG systems offer flexibility in creating context-aware conversations. This allows the chatbot to be both interactive and informative in ways that are not possible using a pure LLM. However, implementing such a system locally also brings several challenges:

- Managing local resource constraints effectively (CPU, memory).
- Ensuring that the file processing and indexing are performant even on larger datasets.

2.6 Implementation Challenges and Solutions

The main challenge was managing different modes (RAG vs non-RAG) efficiently. This was addressed by building separate initialization functions (`init_models_rag` and `init_models_non_rag`), ensuring each mode is self-contained but shares common components such as file handling and memory management.

The file handling process also required optimization, as large files could slow down indexing. Using temporary directories and resetting file pointers after reading helped to mitigate memory issues.

3 User Experience and Functionality

3.1 Usability of the RAG-Enhanced Chatbot

The user interface is intuitive, with clear instructions for toggling modes, uploading files, and controlling parameters like temperature and context length. The chat history is easily manageable, with options to clear or restart conversations.

3.2 Error Handling and Performance

Error handling was implemented for key operations such as file uploads, model execution, and subprocess management. For example, if a user attempts to run a model without proper installation of Ollama, the system returns an informative error message rather than crashing.

3.3 Additional RAG-Specific Features

The RAG system introduces the ability to upload multiple document types and manage them within the same session. This enhances user flexibility and allows for more complex interactions, such as querying across several data sources.