universität wien

Evaluation Approaches for Retrieval Augmented Generation (RAG)

Daniel Petrov - a12028482@unet.univie.ac.at

May 28, 2025, Vienna

Contents

1 Introduction

1.1 Why Are Standalone LLMs Not Good Enough?

Large Language Models (LLMs) such as GPT, LLaMA, and PaLM have demonstrated remarkable capabilities across a variety of natural language processing (NLP) tasks. These models store information within their parameters and rely on statistical correlations learned from massive training corpora. However, they face significant limitations, especially when applied to domain-specific or knowledge-intensive tasks [1, 2].

A key issue is the tendency of LLMs to produce hallucinations, where they confidently generate information that is factually incorrect. McKenna et al. [3] identify two main biases responsible for this behavior:

- **Memorization Bias**: The model over-relies on memorized patterns from the training data, often responding based on previously seen phrases or facts.
- **Statistical Frequency Bias**: The model is more likely to assert information that frequently appeared during training, mistaking frequency for truth.

Furthermore, standalone LLMs operate as black boxes, making it difficult to trace the origin of their outputs or update their internal knowledge base. This lack of transparency and adaptability limits their practical usability, especially in rapidly evolving fields or high-stakes applications.

1.2 Retrieval-Augmented Generation (RAG)

To overcome the limitations of standalone LLMs, the *Retrieval-Augmented Generation* (RAG) framework was introduced. RAG augments generative models with an external, non-parametric memory—typically in the form of a document corpus indexed in a vector database. The architecture combines a retriever, which fetches relevant documents based on a user query, and a generator, which uses the retrieved information to produce a grounded response [1, 2].

By accessing external data at inference time, RAG models benefit from improved factual accuracy, traceability (as sources can be inspected), and the ability to easily update or remove knowledge without retraining the entire model. This modularity also makes them more robust to outdated or domain-specific information.

1.3 Stages

A typical RAG pipeline consists of three main stages: *Indexing*, *Retrieval*, and *Generation* [1].

1.3.1 Indexing

In the indexing stage, heterogeneous knowledge sources—such as PDFs, Markdown files, or knowledge graphs are pre-processed and converted into plain text. The content is split into manageable chunks using strategies like character-based or token-based splitting. These chunks are then encoded into vector representations using embedding models, and stored in a vector store (e.g., FAISS, pgvector) to support efficient similarity-based retrieval.

1.3.2 Retrieval

Given a user query, the retriever encodes it into an embedding and searches for the most semantically similar documents using cosine similarity. Most modern RAG systems use dense retrievers powered by embedding models, although sparse or hybrid retrieval strategies can also be used [2].

To improve retrieval accuracy, techniques such as re-ranking, metadata filtering, or relevance scoring may be applied in a post-retrieval phase [1].

1.3.3 Generation

The generator takes the original user query and the retrieved documents to form an augmented prompt, which is then passed to an LLM to generate a response. This process allows the model to remain grounded in factual evidence and improves the relevance and credibility of its answers.

Additional prompt engineering techniques, such as few-shot examples or explicit instructions, can further steer the generator to use the context appropriately.

1.4 Implementation

The RAG pipeline evaluated in this thesis is implemented using the open-source `R2R` framework[1], which is designed to facilitate end-to-end evaluation of retrieval-augmented generation systems. The complete source code for this project is publicly available[2]. Streamlit-based web interface provides an intuitive chatbot experience for users to interact with the system.

The system supports two main functions: document ingestion and a simple conversational question-answering chatbot. In the ingestion phase, documents are parsed using the `unstructured` library, split into overlapping fragments using a `RecursiveCharacterTextSplitter`, and encoded into dense vectors using the `mxbai-embed-large` embedding model. These vectors are stored in a PostgreSQL database using the `pgvector` extension, with an HNSW index used to speed up approximate nearest neighbor search.

The conversational interface maintains conversation history through R2R's conversation management system, where each message is augmented with computed embeddings to enable semantic similarity matching against previous interactions. This allows the system to retrieve relevant historical context when processing new queries, enhancing response coherence across conversation turns. At query time, the system follows a multi-stage retrieval and generation process:

1. **Query Enhancement**: The user query is potentially augmented with relevant historical context from previous conversation turns using semantic similarity matching
2. **Semantic Retrieval**: The enhanced query is encoded and used to retrieve the top-K most similar document chunks from the vector store using cosine similarity
3. **Re-ranking**: Retrieved chunks undergo a re-ranking process to improve relevance ordering
4. **Response Generation**: A prompt combining the original query, retrieved context, and conversation history is composed and passed to a local LLM served via `Ollama`

The generation process uses configurable parameters including temperature, top-p sampling, and maximum token limits. The system employs streaming responses to provide real-time feedback to users. Custom prompt templates can be selected to guide the LLM's behavior, with explicit instructions to rely solely on the provided context to minimize hallucinations. This implementation follows the Naive RAG architecture as described by Gao et al. [1], incorporating conversation memory and query enhancement mechanisms that extend beyond the basic retrieval-generation paradigm.

At query time, the system encodes the user input, retrieves the top-K most similar chunks, re-ranks them, and composes a prompt that includes both the query and retrieved context. The final response is generated using a local LLM served via `Ollama`, instructed to rely solely on the provided context. This basic Naive RAG structure closely follows the architecture described in the survey by Gao et al. [1].

[1] https://r2r-docs.sciphi.ai/introduction
[2] https://github.com/danielpetrov18/Evaluation-Approaches-for-Retrieval-Augmented-Generation-RAG-

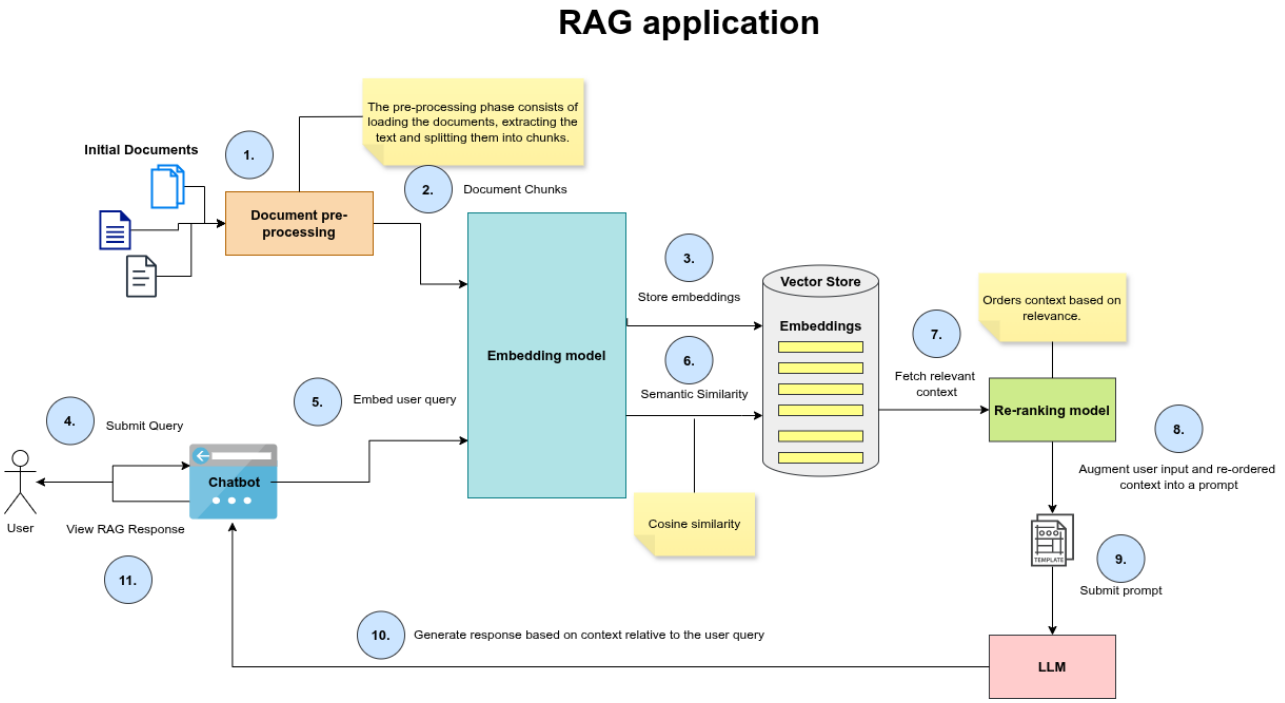

Figure 1: An overview of the implemented RAG pipeline

2 Evaluation

3 Dataset

4 Experiments

5 Conclusions

References

[1] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey, 2024.

[2] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

[3] Nick McKenna, Tianyi Li, Liang Cheng, Mohammad Javad Hosseini, Mark Johnson, and Mark Steedman. Sources of hallucination by large language models on inference tasks, 2023.