AI – Using a Large Language Model (LLM) together with Retrieval-Augmented Generation (RAG) for answering questions on custom data

University of Vienna | Faculty of Computer Science | Bachelor Thesis Presentation

Ivan Khomich

Abstract

This bachelor's thesis focuses on the development of a chatbot application that integrates a large language model (LLM) with Retrieval-Augmented Generation (RAG) to provide accurate and contextually appropriate responses based on custom datasets. The objective is to develop a chatbot that can efficiently retrieve relevant information from uploaded documents and websites and use this data to generate accurate responses.

Motivation and Problem Statement

- **Context:**
  - Use of large language models has advanced beyond generic Q&A into specialized fields.
  - Standard LLMs often struggle with custom, domain-specific, or newly updated info.
- **Retrieval-Augmented Generation (RAG)[8]:**
  - Integrates LLMs with a retrieval mechanism to fetch relevant context from specific datasets.
  - Addresses limitations of standard LLMs, which cannot easily adapt to new information in real time.
- **Local LLM Focus:**
  - Avoids ongoing cloud costs.
  - Enhanced privacy – data stays local.

Project Goals

- **Primary Goal:** Implement a chatbot with a local LLM + RAG for domain-specific Q&A.
- **Key Metrics:**
  - **Accuracy:** Generate factually correct answers using external context.
  - **Efficiency:** Handle large datasets with minimal latency.
  - **Adaptability:** No continual retraining required to handle new or updated data.
- **Scope:**
  - RAG retrieves *specific* doc segments.
  - Not designed for complex tasks like extensive summarization or multi-step reasoning.

universität wien

Process and Methods

- **RAG Pipeline:**
  - Uses vector database to fetch relevant context.
  - LLM uses that context to craft final response.
- **ReActAgent[13]:**
  - Dynamically decides how to solve query.
  - Possible tools: web search, RAG-based doc retrieval, math solver.
- **Streamlit UI[11]:**
  - Upload documents, clean db, and query the chatbot.

Related Work

Smaller Individual Works (Guides/Tutorials)

- Mostly illustrate basic usage or toy demos.
- Lack of robust GUIs or multi-dataset integration.
- Overly simplistic, using only default prompts.

Related Work: Comparison of RAG Implementations

Approach	Strengths	Weaknesses
WebGPT[9]	High accuracy, integrates search	Prone to hallucinations, retrieval of low-quality sources
Haystack[1]	Handles multiple datasets, scalable	Slow on large datasets
LangChain[3]	Easy to use, widely adopted	Poor text splitting, simplistic prompt engineering
This Thesis	Runs locally, high privacy	Limited testing, no multi-turn retrieval

Table: Comparison of Different RAG Implementations

Related Work

The new generation of models—GPT-4o, o1, and o3, Claude 3.5s —are “better” than older RAG implementations because they:

- **Think internally**: They have built-in chain-of-thought reasoning that solves complex tasks step-by-step without needing heavy external prompt engineering.
- **Handle more data**: Their greater context windows and multimodal capabilities allow them to ingest and process much larger inputs.
- **Improve safety**: Advanced techniques like deliberative alignment help them generate safer, more reliable outputs.
- **Excel in challenging domains**: Their performance benchmarks shows they outstrip what can be achieved by combining a basic LLM with a retrieval layer.

universität wien

Retrieval-Augmented Generation (RAG)

- RAG = LLM + document-retrieval component.
- Steps:
  1. Retrieve relevant context from data store.
  2. Provide context to the LLM for a more accurate response.

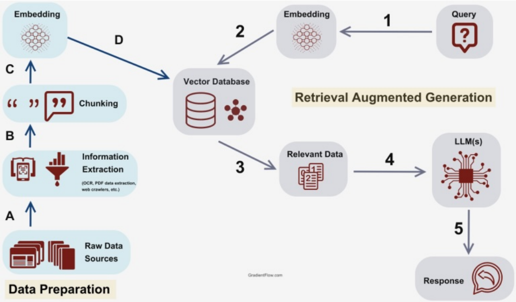


Figure: RAG workflow[6]

Large Language Models (LLMs)

- **Examples:** Llama, GPT, etc.
- **Focus here:** Llama 3.1[2] + LlamaIndex ReAct agent[4].
- **Agent-based approach:**
  - Calls external tools for tasks like web search or math solver.

Ollama[10]

- **Open-source platform** for running LLMs locally.
- **Benefits**:
  - Free, minimal license issues.
  - Full control over data, offline usage.
  - Simplifies local LLM integration.

Embeddings & Vector Database

- **Embeddings:**
  - High-dimensional vectors capturing semantic context, this project uses **bge-base-en-v1.5**[5].
  - Enable efficient similarity search for text, images, or audio.
- **Vector Database:**
  - Stores and queries these high-dimensional vectors.
  - **ChromaDB[12]:**
    - Used for storing embeddings + metadata.
    - Key for text-based similarity search in LLMs.

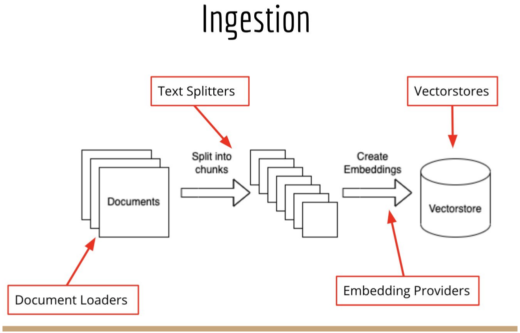


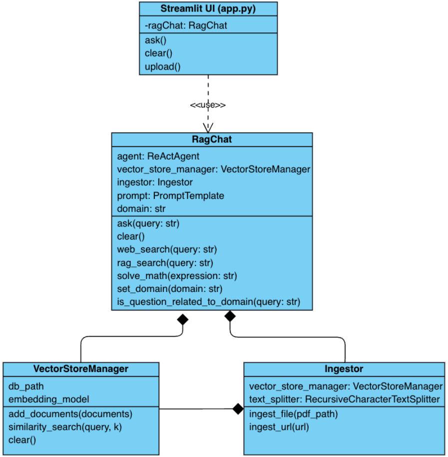
Figure: Data ingestion and storage process [7]

Data Preparation

- **Uploaded PDFs:**
  - `PyPDFLoader` for text extraction.
  - `RecursiveCharacterTextSplitter` for chunking.
- **Webpages:**
  - `requests` + `BeautifulSoup` for parsing.
- **Embeddings in Chroma**:
  - Each chunk vectorized and stored for fast retrieval.

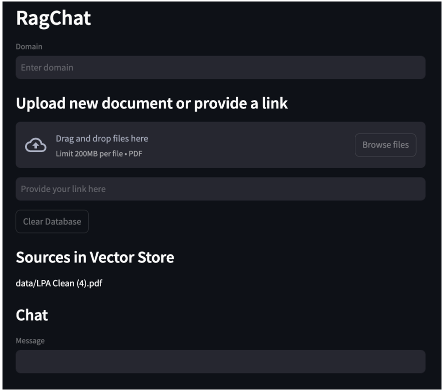
Implementation Overview

- **UI (Streamlit)** for user interaction.
- **LlamaIndex ReActAgent** for orchestration.
- **Document Parsing & Vector DB** for retrieval.



UI (Streamlit)

- Web-based interface:
  - Upload documents, input URLs.
  - Interact with chatbot Q&A.
  - Clear or update database.



LlamaIndex ReActAgent

- **Process:**

1. User Query
2. Agent decides whether to:
   - Retrieve from local docs.
   - Use web search.
   - Use math solver.
3. Synthesizes final answer.

Document Retrieval and Storage

- Embeddings stored in `Chroma`.
- Agent queries top matching chunks for local context.
- Chunks inserted into LLM prompt for context-aware answers.

Data Ingestion Pipeline

1. **Parse Documents:** PDFs or web sources.
2. **Split Text:** `RecursiveCharacterTextSplitter`.
3. **Store Embeddings:** $\rightarrow$ vector DB (Chroma).

Repository

- Full implementation with RAG + local LLM.
- Detailed README for setup.

Ref: `https://github.com/grippvh/ragChat`

Test Coverage

- Focus on verifying correct retrieval + context usage.
- **Challenge:** LLM outputs can vary in wording.
- **Strategy:**
  - Ask the model, then re-ask it to validate its answer against expected response.
  - Negative test cases ensure incorrect answers are flagged.
- **No 100% determinism:** LLM variability can produce different valid answers.

Observations

- Combining LLM + RAG can be tricky (embedding quality, chunking).
- **Naive RAG:**
  - Works on small docs.
  - Degrades with larger, complex data if not optimized.
- **Key Factor:**
  - Embeddings heavily influence retrieval quality.
  - Prompt design also matters.

Performance Example

- **Experiment Setup:** As an example, the Constitution of the Republic of Belarus was used — a document unlikely to be included in the training dataset of large language models. The experiment involved querying different models with the question, "What does Article 4 of the Belarus Constitution state?"
- **Without RAG:** LLaMA 3.1 generated a factually correct response, but it was irrelevant to the query, demonstrating the model's inability to retrieve specific information from external sources. GPT-4 provided a more accurate answer but required a web search to locate the correct information.
- **With RAG:** LLaMA 3.1, enhanced with RAG, retrieved the exact text of Article 4 directly from the document. This result highlights the advantage of integrating a retrieval mechanism for domain-specific or fresh datasets.
- **Conclusion:** RAG proved essential for accurately answering questions about domain-specific content not included in the training data of standard LLMs.

universität wien

Summary of Findings

- **Accuracy:** Increases when relevant context is retrieved.
- **Limits:** Not ideal for broad tasks like full summarization.
- **Potential:** Additional modules (knowledge graphs, rerankers) could improve coverage.

Lessons Learned

- **Better embeddings** significantly improve retrieval accuracy.
- **RAG is best for factual queries,** less so for deeper or multi-doc comprehension.
- **The Impact of Configurations:** Fine-tuning parameters such as chunk size, overlap size, and prompt design can significantly influence the system’s effectiveness, highlighting the importance of systematic experimentation.

Future Improvements

- Smarter query rewriting and rerankers.
- Integrating knowledge graphs.
- Multi-agent approaches for more complex tasks.

Conclusion

- **Local LLM + RAG:** Real-time adaptivity, data privacy, better domain Q&A.
- Even simple RAG improves domain-specific Q&A.
- Fine-tuning not always necessary; data retrieval may be often enough.

References I

- Deepset AI.
  Haystack: An open source nlp framework for question answering, retrieval, and document ranking, 2021.
- Meta AI.
  Meta llama 3.1: Advancing open-source ai, 2024.
- Harrison Chase.
  Langchain: A framework for developing applications powered by large language models, 2023.
- LlamaIndex Documentation.
  React agent api reference.
- Hugging Face and BAAI.
  bge-base-en-v1.5.

References II

Gradient Flow.
Techniques, challenges, and future of augmented language models, 2023.

LangChainAI.
Langchain ai tweet on rag implementation.

Patrick Lewis, Ethan Perez Oguz, Rami Rinott, Sebastian Riedel, and Pontus Stenetorp.
Retrieval-augmented generation for knowledge-intensive nlp tasks.
Advances in Neural Information Processing Systems, 2020.

Reiichiro Nakano, Jacob Hilton, Long Ouyang, et al.
Webgpt: Browser-assisted question-answering with human feedback.
arXiv preprint arXiv:2112.09332, 2021.

References III

- Ollama.
  Ollama: Run large language models locally.
- Streamlit Team.
  Streamlit Documentation, 2021.
- Nidhi Worah.
  Chroma db introduction, 2024.
- Shinnong Yao, Weijie Zhao, Dongsheng Yu, Yuan Cao, and Zexuan Zhao.
  React: Synergizing reasoning and acting in language models.
  Advances in Neural Information Processing Systems, 2022.