

Retrieval-Augmented Generation (RAG) System using Groq, LangChain, and ChromaDB

January 16, 2026

1 Introduction

1.1 What is Retrieval-Augmented Generation (RAG)?

Retrieval-Augmented Generation (RAG) is a hybrid natural language processing approach that combines information retrieval with large language models (LLMs). Instead of relying solely on a model's parametric memory, RAG dynamically retrieves relevant external documents and injects them into the prompt context before generating a response.

This approach significantly improves:

- Factual accuracy
- Domain adaptability
- Transparency of answers
- Reduction of hallucinations

RAG is especially important for applications such as question answering over documents, research assistance, enterprise knowledge bases, and document summarization.

1.2 Overview of Groq, LangChain, and ChromaDB

Groq provides ultra-fast inference for large language models using custom LPU hardware. It enables low-latency responses suitable for interactive RAG systems.

LangChain is a framework that simplifies the orchestration of LLMs, retrievers, and tools. It provides abstractions for document loaders, text splitters, retrievers, and chains.

ChromaDB is an open-source vector database designed for storing and querying embeddings efficiently. It supports metadata-based filtering and persistent storage, making it ideal for RAG pipelines.

2 Environment & LLM Setup

2.1 Groq Model Configuration

The Groq API is configured using an API key stored securely in a `.env` file.

```
GROQ_APIKEY=your_api_key_here
```

The Groq client is initialized in Python as follows:

```
from groq import Groq
import os

client = Groq(api_key=os.getenv("GROQ_APIKEY"))
```

2.2 First Successful LLM Call

A sample prompt was sent to validate the setup:

```
response = client.chat.completions.create(
    model="llama-3.1-8b-instant",
    messages=[{"role": "user", "content": "What is -RAG?"}]
)
print(response.choices[0].message.content)
```

3 Tool Development

3.1 Web Crawler Tool

The Web Crawler tool extracts clean textual content from a given URL using HTML parsing. It removes scripts, styles, and irrelevant markup.

Input: Web URL

Output: Extracted clean text

```
(venv) PS C:\Users\naikj\Documents\rag> python main.py
`langchain-chroma package and should be used instead. To use it run `pip install -U `langchain-chroma` and import as `from `langchain_chroma import Chroma``.
vectordb = Chroma(
Source already ingested. Skipping.

RAG System with Groq (type 'exit' to quit)

You: What is Retrieval Augmented GEneration?

Assistant: Retrieval-augmented generation (RAG) enhances large language models (LLMs) by incorporating an information-retrieval mechanism that allows models to access and utilize additional data beyond their original training set.

You: What is chunking in that?

Assistant: Chunking involves various strategies for breaking up the data into vectors so the retriever can find details in it. Different data styles have patterns that correct chunking can take advantage of. Three types of chunking strategies are:

1. Fixed length with overlap
2. Syntax-based chunks
3. File format-based chunking.

You: 
```

Figure 1: Web Crawler Tool Execution and Extracted Content

3.2 Sample Web Crawl Run

- URL: Mayo Clinic Diabetes Page
- Extracted structured medical content
- Successfully chunked and ingested into ChromaDB

3.3 Research Paper Scraper Tool

The Research Paper Scraper extracts text from PDF research papers using pdfplumber with fallbacks to PyPDF2. Noise such as references, tables, and figures is removed.

Handled Sections:

- Abstract
- Introduction
- Conclusion (when available)

```

>You: exit
% (venv) PS C:\Users\naikj\Documents\rag> python main.py
C:/Users/naikj/Documents/rag/utils/vector_store.py:8: LangChainDeprecationWarning: The class `HuggingFaceEmbeddings` was deprecated in LangChain 0.2.2 and will be removed in 1.0. An updated version of the class exists in the `langchain-huggingface` package and should be used instead. To use it run `pip install -U langchain-huggingface` and import as `from langchain_huggingface import HuggingFaceEmbeddings`.
C:/Users/naikj/Documents/rag/utils/vector_store.py:12: LangChainDeprecationWarning: The class `Chroma` was deprecated in LangChain 0.2.9 and will be removed in 1.0. An updated version of the class exists in the `langchain-chroma` package and should be used instead. To use it run `pip install -U langchain-chroma` and import as `from langchain_chroma import Chroma`.
RAG System with Groq (type 'exit' to quit)

You: What is the situation overview described in this report?

Assistant: The situation overview described in this report is a global update on COVID-19 cases, as of April 23, 2020. The report categorizes countries/territories/areas into different classifications based on the number of cases and transmission patterns. The classifications include:
- No cases (not shown in the table)
- Sporadic cases (countries/territories/areas with one or more cases, imported or locally detected)
- Clusters of cases (countries/territories/areas experiencing cases, clustered in time, geographic location, and/or by common exposures)
- Community transmission (countries/area/territories experiencing larger outbreaks of local transmission)

The report provides a breakdown of the number of cases, deaths, and transmission days for various countries/territories/areas in the Western Pacific region, including China, Japan, the Republic of Korea, Singapore, the Philippines, Australia, Malaysia, New Zealand, Vietnam, Brunei Darussalam, Cambodia, Mongolia, and the Lao People's Democratic Republic.

```

Figure 2: Research Paper Scraper Processing a PDF Document

3.4 Sample Output

- Clean abstract text
- Section-aware extraction
- Improved readability for chunking

4 Vector Database (ChromaDB)

4.1 Embedding Generation Approach

Sentence-level embeddings are generated using:

- `sentence-transformers/all-MiniLM-L6-v2`

Each text chunk is converted into a dense vector representation capturing semantic meaning.

4.2 Document Storage and Retrieval

Documents are stored in ChromaDB as:

- Page content (chunk text)
- Metadata (source URL or PDF link)

At query time, similarity search retrieves the top-k most relevant chunks using cosine similarity.

5 RAG Implementation

5.1 System Architecture

User Query → Retriever (ChromaDB) → Relevant Chunks → Groq LLM → Answer

5.2 RetrievalQA Chain

```
def ask_rag(query):
    docs = retriever.get_relevant_documents(query)
    context = "\n".join([doc.page_content for doc in docs])
    prompt = f"Context:\n{context}\n\nQuestion:\n{query}"
    return generate_answer(prompt)
```

5.3 Example Queries and Responses

Query: What problem does the Transformer paper aim to solve?

Response: The paper addresses inefficiencies of recurrent and convolutional models by introducing an attention-only architecture.

Query: What are symptoms of high blood sugar?

Response: Feeling tired and weak is a commonly reported symptom.

6 Conclusion

6.1 Lessons Learned

- Chunk quality directly impacts answer relevance
- PDF parsing varies greatly by document source
- Metadata filtering is crucial for source control
- RAG significantly improves factual grounding

6.2 Limitations and Future Improvements

- Multi-agent RAG for reasoning + retrieval separation
- Streaming responses for improved UX
- Hybrid search (keyword + vector)