# Retrieval-Augmented Generation for Semantic Querying of PDF Documents Using Open-Source Language Models

Ansh Bharghava
*Dept. of Computer Science and Engineering*
*Sir M. Visvesvaraya Institute of Technology*
Bengaluru, Karnataka

Brinda B V
*Dept. of Computer Science and Engineering*
*Sir M. Visvesvaraya Institute of Technology*
Bengaluru, Karnataka

Mrs. Itishree Barik
*Dept. of Computer Science and Engineering*
*Sir M. Visvesvaraya Institute of Technology*
Bengaluru, Karnataka

Arunita Sahu
*Dept. of Computer Science and Engineering*
*Sir M. Visvesvaraya Institute of Technology*
Bengaluru, Karnataka

Nandini Kumari
*Dept. of Computer Science and Engineering*
*Sir M. Visvesvaraya Institute of Technology*
Bengaluru, Karnataka

***Abstract* — Due to their dependence on static, pre-trained knowledge, large language models (LLMs) frequently experience hallucination, producing outputs that appear confident but are factually incorrect. This restriction is lessened by retrieval-augmented generation (RAG), which grounds responses in outside context. A document-based RAG system that uses the open-source reasoning model DeepSeek and is locally deployed using Ollama is presented in this paper. A Chroma vector database is used to store the PDF documents that users upload after they have been parsed, segmented, and embedded using LangChain. When a query is received, the system uses DeepSeek to produce grounded responses and retrieve semantically relevant content. The system may optionally call the web-based search API Tavily if no appropriate context is found. Streamlit, which provides an easy-to-use interface for document upload, parameter tuning, and chat interaction, is used to deploy the application. The system enhances reliability and decreases hallucinations in knowledge-intensive tasks by integrating contextual retrieval, local inference, and optional fallback.**

**Keywords — LLMs, Hallucination, RAG, DeepSeek, Ollama, Local Deployment, Vector Database.**

## I. INTRODUCTION

Large Language Models (LLMs) have remarkable capacity for natural language understanding and generation. But their reliance on static pre-trained corpora leads to a terrible flaw: hallucination. It is a phenomenon of generating coherent yet factually incorrect or unverifiable text. In high-stakes domains like law, medicine, and scholarship, such mistakes can undermine trust and decrease value.

Retrieval-Augmented Generation (RAG) solves this problem by augmenting large language models (LLMs) with dynamic access to external knowledge at runtime. Rather than depending solely on internal parameters, RAG models fetch relevant documents from a knowledge base at runtime and generate their response based on this contextual information. Hallucination is reduced, fact accuracy is improved, and domain adaptation is enabled.

In this paper, we use a local RAG pipeline that integrates open-source components: DeepSeek for reasoning, Ollama for optimized model serving, LangChain for orchestration, and Chroma as a vector store. Users can upload PDFs, which are parsed, chunked, and added to a searchable vector database. When a query is received, the system retrieves semantically similar content and sends it to the LLM for the grounding of responses. The system also has Tavily, a real-time web search API, for use as a fallback in situations where there is no internal context of relevance.

The system is implemented via a Streamlit interface to enable easy interaction, parameter adjustment (i.e., number of retrievals and threshold for similarity), and stored chat history. This renders the tool appropriate for knowledge-intensive tasks where contextual correctness and responsiveness take centre stage.

## II. RELATED WORKS

### A. *Overview of existing solutions*

The following research studies and advancements in Retrieval-Augmented Generation (RAG) systems were reviewed to understand various methods and technologies aimed at improving large language models (LLMs) and mitigating hallucinations. These studies highlight different strategies for integrating external knowledge into generative models.

- Sonia Vakayil, D. Sujitha Juliet et al. (2024) in "RAG-Based LLM Chatbot Using Llama-2" employed Retrieval-Augmented Generation (RAG) with the Llama-2 model to build an empathetic chatbot for sexual harassment victims, combining document retrieval with LLM generation for accurate and sensitive responses.
- Oscar Cederlund, Sadi Alawadi et al. (2024) in "LLMRAG: An Optimized Digital Support Service using LLM and Retrieval-Augmented Generation" applied RAG to automate solution suggestions in an IT service desk, integrating LLMs with ticket-specific retrieval to assist technicians, with 38.4% of generated solutions being retained during a two-week trial.
- Büşra Tural et al. (2024) in *"Retrieval-Augmented Generation (RAG) and LLM Integration"* explored the integration of RAG architecture with LLMs to overcome the limitations of traditional keyword-based Information Retrieval. By enabling LLMs to dynamically retrieve semantically relevant information from external sources, their approach aimed to generate more accurate and context-aware responses for complex, information-heavy tasks.

- Yunfan Gao et al. (2024) in "Retrieval-Augmented Generation for Large Language Models: A Survey" provided a comprehensive review of RAG systems, categorizing them into Naive, Advanced, and Modular paradigms. The study dissected the RAG pipeline into retrieval, generation, and augmentation components, discussing innovative techniques in each. It also proposed evaluation benchmarks and addressed ongoing challenges like hallucinations and domain adaptation.

- Omrani et al. (2025) in "Hybrid Retrieval-Augmented Generation Approach for LLMs Query Response Enhancement" introduced a novel hybrid RAG framework. This framework combines Sentence-Window and Parent-Child methodologies with a new re-ranking mechanism to improve the query response capabilities of LLMs. The authors demonstrated through rigorous evaluation against benchmark datasets that their hybrid model outperforms existing state-of-the-art RAG techniques in terms of accuracy, relevance, and faithfulness to the source material. [1] The study highlights the potential of hybrid RAG models for enhancing the interaction between LLMs and external knowledge.

- M. T. Huang et al. (2024) in "Scalable Retrieval-Augmented Generation Systems for Open-Domain Question Answering" [4] proposed an open-domain Q&A system using RAG, with embedded documents stored in vector databases and queried for semantically relevant content. Their system is designed to scale for large-scale datasets and improve response accuracy.

- A. T. Nguyen et al. (2024) in "Fine-Tuning Retrieval-Augmented Models with Real-Time Document Embedding" focused on fine-tuning models to integrate real-time document embeddings. Their approach combined traditional retrieval methods with deep learning models for embedding, demonstrating how fine-tuning improves document relevance in RAG applications.

- R. S. Patel et al. (2023) in "Improving Retrieval-Augmented Generation Models for Long-Form Text Generation" demonstrated a retrieval approach aimed at long-form text generation tasks. Their system showed promising results in extending the context of document retrieval to better generate comprehensive and contextually grounded responses.

### B. *Gaps in existing solutions*

- Inadequate Adaptation to Complex Knowledge Structures and Long-Form Text Generation and obsolete knowledge: Research often presents complex, multi-document structures. Current RAG methods struggle to fully utilize relationships within these structures for coherent, detailed long-form text generation.

- Absence of Specialized Benchmarks for Niche Applications: Existing RAG benchmarks are too general, as highlighted by Gao et al. (2023). Specialized evaluation frameworks are needed for niche scientific and technical tasks like academic QA and summarization.

- Potential for Language and Contextual Bias: While the presented papers primarily focus on English language applications, the broader issue of language and contextual bias in RAG systems warrants consideration. Ensuring inclusivity for non-English and low-resource languages remains a challenge for the widespread adoption of these technologies.

- Limited Explainability in Retrieval and Generation Processes: Understanding why specific documents are retrieved and how the final response is generated remains a challenge in many RAG implementations. Enhancing the transparency of these processes would improve user trust and facilitate system refinement.

- Insufficient User Customization Capabilities: Providing end-users, particularly those without deep technical knowledge, with greater control over the retrieval scope, document sets, and response style is an area for improvement in current RAG-based LLM solutions.

- Embedding Quality and Retrieval Relevance: The effectiveness of RAG systems heavily relies on the quality of document embeddings. Poor embedding quality can lead to the retrieval of irrelevant information, negatively impacting the accuracy of the generated responses.

## III. METHODOLOGY

The goal of the project is to create a Retrieval-Augmented Generation (RAG) system that enriches the functionality of large language models (LLMs) by blending external document-based knowledge. The system is created to be run locally to provide context-specific and accurate replies to user questions by retrieving pertinent information from uploaded PDFs and passing it through an open-source LLM. The system also includes a fallback module to retrieve live web results in case there is no local content.

### A. *System Design and Architecture*

The methodology begins with designing the overall architecture of the RAG pipeline, with individual modules managing document processing, embedding, retrieval, model inference, and user interaction. The system is modular and designed to allow flexibility, scalability, and transparency in both internal processing and user interaction. Following are the core components of the system:

- PDF Loader (PyPDFLoader): Reads and splits the uploaded document into smaller, manageable chunks for downstream semantic processing.

- Text Splitter (RecursiveCharacterTextSplitter): Segments the document into overlapping text chunks, preserving semantic coherence across splits and enhancing retrieval quality.

- Embedding Model (OllamaEmbeddings): Converts text chunks into dense vector representations using an LLM running locally via Ollama. This facilitates high-dimensional semantic search.

- Vector Store (Chroma): Acts as a persistent memory layer that stores the generated embeddings and supports fast, similarity-based retrieval using cosine distance metrics.

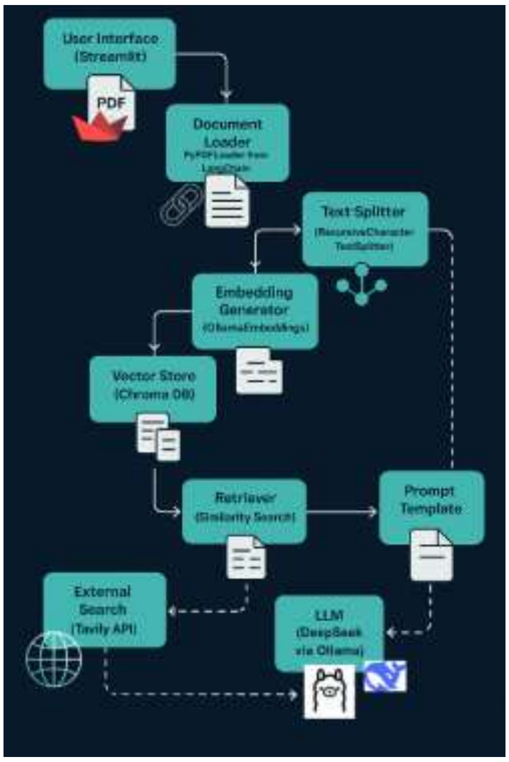
- Retriever: Searches the vector store to fetch the top-k most relevant text chunks corresponding to the user's query.

- Tavily Retriever: A fallback mechanism triggered when the local retriever fails to meet the similarity threshold, enabling real-time web search and reducing dependence on local data alone.

- LLM (DeepSeek via Ollama): Processes both the user query and the retrieved context to generate a concise, context-aware response. The model runs locally, ensuring privacy and control over inference.

- LangChain: Orchestrates the end-to-end pipeline by managing prompt construction, conditional retrieval logic, and response formatting. It ensures seamless interaction between the modules.

- Streamlit Frontend: Offers an intuitive user interface for uploading documents, entering queries, visualizing system responses, and adjusting key parameters such as similarity thresholds or number of returned documents.


User Interface (Streamlit)
PDF
Document Loader PyPDFLoader from LangChain
Text Splitter (RecursiveCharacterTextSplitter)
Embedding Generator (OllamaEmbeddings)
Vector Store (Chroma DB)
Retriever (Similarity Search)
Prompt Template
External Search (Tavily API)
LLM (DeepSeek via Ollama)

*B. Development Phases*

Phase 1: Document Ingestion and Preprocessing
The system starts with the local ingestion of PDF documents provided by the user. Using PyPDFLoader from LangChain, the textual content is extracted from each document. To maintain contextual integrity, the text is segmented into overlapping chunks with RecursiveCharacterTextSplitter. This method ensures that semantically connected information remains linked across chunks, improving the quality of later retrieval.

Phase 2: Embedding Generation and Vector Storage
Each text chunk is transformed into a high-dimensional vector representation using the OllamaEmbeddings model, which operates locally. These vectors are stored in a persistent Chroma vector database. This phase forms the backbone of semantic search functionality, enabling efficient and accurate retrieval of relevant content when a user query is issued.

Phase 3: Query-Based Retrieval
When a user submits a query, the system uses a retriever to perform a similarity search across the stored embeddings in the Chroma vector database. The retriever identifies the top-k most semantically similar text chunks based on cosine similarity. If the similarity score of all retrieved chunks falls below a predefined threshold, the system initiates a fallback mechanism that calls the Tavily web search API. This ensures access to up-to-date external knowledge when local data is insufficient.

Phase 4: LLM Inference and Answer Generation
After retrieving the relevant chunks, a prompt is dynamically formatted by inserting the retrieved context and the user query into a predefined template. This prompt is passed to the locally running DeepSeek language model via Ollama. The model then generates a grounded, context-aware response. Prompt length and token usage are managed to avoid truncation or hallucination, and the output is structured for clarity and relevance.

Phase 5: User Interface and Customization
The entire system is deployed via a Streamlit application that provides a clean and interactive front end. Users can upload PDF files, enter queries, and view generated responses in real-time. Additionally, the interface includes controls to adjust retrieval parameters such as top-k values and similarity thresholds, toggle between local and external search modes, and view the source context and system logs for transparency and debugging.

*C. Performance Evaluation*

The system's effectiveness was evaluated based on the following criteria:

- Response Accuracy: Measured by human validation against source documents for factual correctness.
- Relevance Score: Calculated using cosine similarity between the query and retrieved context chunks.
- Latency: Measured in milliseconds from query input to answer generation, including fallback time when Tavily is triggered.

Metrics were collected under controlled conditions using a curated set of standardized PDF documents containing varied technical and academic content. Evaluation was conducted across three configurations: with only local retrieval, with fallback search enabled, and using DeepSeek versus alternative models for comparative purposes.

*D. System Implementation*

The final system was deployed as a local web application using Streamlit, integrating all core RAG components seamlessly.
Key features included:
• PDF upload and real-time parsing interface
• Chat-style interaction with adjustable retrieval parameters (Top-K, similarity threshold)
• Toggle for enabling or disabling web fallback (Tavily API)
• Context display for transparency in answer generation
• Downloadable chat history for user reference

The backend was made modular for better scalability and testability, with all crucial processes—i.e., loading, embedding, vector storage, retrieval, and generation—kept as standalone, callable modules. The application was cross-platform packaged for deployment, with comprehensive documentation for installation, upkeep, and parameter tuning.

## IV. RESULTS AND DISCUSSIONS

To ensure the performance of the proposed Retrieval-Augmented Generation (RAG) system, we thoroughly assessed it on a wide variety of academic and technical PDFs. The system was evaluated for its ability to generate semantically correct and contextually relevant responses using locally cached document embeddings as well as external fallback configurations.

### *A. Relevance and Accuracy of Response*

The system always returned correct, semantically grounded responses when there was applicable content present in the documents that were uploaded. DeepSeek, accessed locally through Ollama, excelled at long-context reasoning and generation of fact responses. In situations where document context was lacking, Tavily's web search fallback could insert extra grounding with good relevance.

### *B. User Experience*

Streamlit interface facilitated effortless document uploading, parameter adjustment, and real-time chat-based conversation. The inclusion of chat history, top-k retrieval controls, and context visibility contributed to a user-friendly experience. The debug visibility feature promoted system transparency further.

### *C. Limitations*

One of the major downsides of the system is the CPU-based high latency of execution, which can have negative impacts on responsiveness, particularly in time-sensitive applications. In addition, long or complex queries can sometimes be truncated because of prompt length limits, thus affecting completeness of responses in some instances. While the web fallback feature via Tavily enhances coverage, it adds infrequent response time and accuracy variability, especially in cases where API rate limits are hit or when network connectivity is poor.

### *D. Future Enhancements*

To further optimize system performance and user interaction, some future extensions are suggested. Utilization of the model on GPU-enabled infrastructure, or compatibility with other optimized inference engines like ONNX Runtime or TensorRT, would have a significant impact on reducing latency and increasing throughput. Support for multi-document querying and user sessions persistence would improve scalability and interactivity. Fine-tuning the DeepSeek model on domain-specific corpora can further improve response accuracy in domain-specific use cases. Additionally, the inclusion of user feedback loops and confidence scoring in replies can enhance trust and transparency in practical applications.

## REFERENCES

[1] S. Vakayil, D. S. Juliet, Anitha. J, and S. Vakayil, “RAG-Based LLM Chatbot Using Llama-2,” *2024 5th International Conference on Intelligent Computing and Communication (ICICC)*, pp. 1803-1807, 2024.

[2] O. Cederlund, S. Alawadi, and F. M. Awaysheh, “LLMRAG: An Optimized Digital Support Service using LLM and Retrieval-Augmented Generation,” *2024 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pp. 1064-1068, 2024.

[3] B. Tural, Z. Örpek, and Z. Destan, “Retrieval-Augmented Generation (RAG) and LLM Integration,” *2024 7th International Conference on Computer Science and Engineering (UBMK)*, pp. 537-540, 2024.

[4] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, “Retrieval-Augmented Generation for Large Language Models: A Survey,” *arXiv preprint arXiv:2312.10997*, 2023.

[5] T. T. Procko and O. Ochoa, “Graph Retrieval-Augmented Generation for Large Language Models: A Survey,” *arXiv preprint arXiv:2310.14253*, 2023.

[6] J P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Y. Wang, and S. Riedel, "Retrieval-augmented generation for knowledge-intensive NLP tasks," Advances in Neural Information Processing Systems, vol. 33, pp. 9459–9474, 2020.

[7] G. Izacard and E. Grave, "Leveraging passage retrieval with generative models for open domain question answering," arXiv preprint arXiv:2007.01282, 2020.

[8] P. Omrani, A. Hosseini, K. Hooshanfar, Z. Ebrahimian, R. Toosi, and M. A. Akhaee, “Hybrid Retrieval-Augmented Generation Approach for LLMs Query Response Enhancement,” *32nd Iranian Conference on Electrical Engineering (ICEE)*, pp. 115-120, 2025.