ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26oct1167

A Survey on Retrieval-Augmented Generation for Multi-PDF Question Answering Systems

Survey Report on the Implemented RAG System

P. S. Togrikar[1]; Tejas Pawar[2]; Akshay Malgunde[3]; Mayureshwar Raychure[4]

[1]Professor
[1;2;3;4]Department of Electronics and Telecommunication Engineering S. B. Patil College of Engineering

Publication Date: 2026/10/31

Abstract: Retrieval-Augmented Generation (RAG) effectively separates knowledge retrieval from generation for private document question answering. This study explores the design space of document-grounded QA and details a multi-PDF RAG system. The pipeline uses PyMuPDF for text extraction with OCR fallback, BAAI/bge-small-en-v1.5 for dense embeddings, and ChromaDB for storage. Semantically similar chunks are retrieved via cosine distance to generate answers using a local Llama 3.2:3B model via Ollama. Built with React/Vite and FastAPI, the system supports document management and query APIs. The paper reviews RAG methodologies—including hybrid retrieval, reranking, and GraphRAG—while maintaining a local, modular architecture optimized for consumer-grade hardware. Finally, it evaluates the system's strengths and limitations to identify future research directions.

Keywords: *Retrieval-Augmented Generation; Multi-PDF Question Answering; Document-Grounded QA; Dense Retrieval; Vector Databases; OCR; Local LLMs.*

How to Cite: P. S. Togrikar; Tejas Pawar; Akshay Malgunde; Mayureshwar Raychure (2026) A Survey on Retrieval-Augmented Generation for Multi-PDF Question Answering Systems. *International Journal of Innovative Science and Research Technology*, 11(10), 2336-2342. https://doi.org/10.38124/ijisrt/26oct1167

I. INTRODUCTION

Large language models (LLMs) can generate fluent answers but may rely on information stored in model parameters rather than the documents a user wants to inspect. This creates practical problems when the knowledge source is private, newly uploaded, domain-specific, or requires traceable evidence. Retrieval-Augmented Generation addresses this limitation by retrieving relevant passages from an external knowledge source and supplying them to the generator at inference time [1]. Survey literature commonly organizes RAG into retrieval, augmentation, and generation stages, with further distinctions between naive, advanced, and modular RAG systems [2], [3].

The project considered in this survey focuses on a narrower but highly useful problem: question answering over one or more user-uploaded PDFs. Instead of sending complete documents to an LLM, the system converts PDF content into searchable chunks and retrieves only the passages relevant to a question. The resulting context is then passed to a local Llama 3.2:3B model with a strict document-only prompt. Source metadata records the filename, page number, chunk index, similarity, and distance, enabling page-aware citations in the user interface.

The project intentionally keeps the architecture unchanged: React + Vite frontend, FastAPI backend, PyMuPDF/OCR document processing, paragraph-aware chunking, BGE embeddings, persistent ChromaDB, and Ollama-hosted Llama 3.2:3B. This survey therefore evaluates the selected technologies as an integrated document-QA pipeline rather than proposing a replacement architecture. The objective is to understand why these components are appropriate, how they relate to research in RAG, and where the present implementation can be improved while preserving the same system boundaries.

- *Need for Rag in Multi-PDF Question Answering*

Manual reading becomes inefficient when a user must repeatedly search several long PDFs for specific facts, definitions, dates, tables, or comparisons. A full-document prompt is also undesirable because it increases context size and may expose irrelevant material to the generator. Retrieval addresses this by selecting a small evidence set before generation. Dense passage retrieval research demonstrated that learned vector representations can be effective for finding semantically relevant passages [4], while sentence-embedding research established efficient cosine-similarity search over semantically meaningful representations [5].

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26oct1167

The multi-PDF setting introduces additional requirements. The system must preserve document identity, page number, and chunk boundaries; support multiple files in a common searchable collection; allow optional filtering to a selected document; and distinguish an unsupported question from a question for which evidence exists. These requirements motivate metadata-aware vector storage and a similarity threshold rather than relying only on a fixed top-k result.

- Document grounding: answers are generated from retrieved PDF context instead of unrestricted model knowledge.
- Multi-document retrieval: chunks from multiple PDFs can coexist in one persistent collection.
- Traceability: source filename, page number, chunk index, and relevance are retained.
- Scanned-document support: OCR provides a fallback when normal PDF text extraction is insufficient.
- Local execution: embeddings and generation can run locally, reducing dependence on external inference APIs.
- Controlled generation: the prompt instructs the LLM to return a fixed not-found response when evidence is insufficient.

II. RAG ARCHITECTURE & COMPONENTS

The implemented architecture follows a conventional retrieve-then-generate pipeline while adding page-aware PDF processing and OCR. The frontend communicates with a FastAPI REST backend. Uploaded files are stored locally, extracted page-by-page, cleaned, chunked, embedded, and indexed in a persistent ChromaDB collection. At query time, the question is embedded using the same embedding model, similar chunks are retrieved, low-relevance candidates are filtered, and the selected context is sent to a local Llama model.

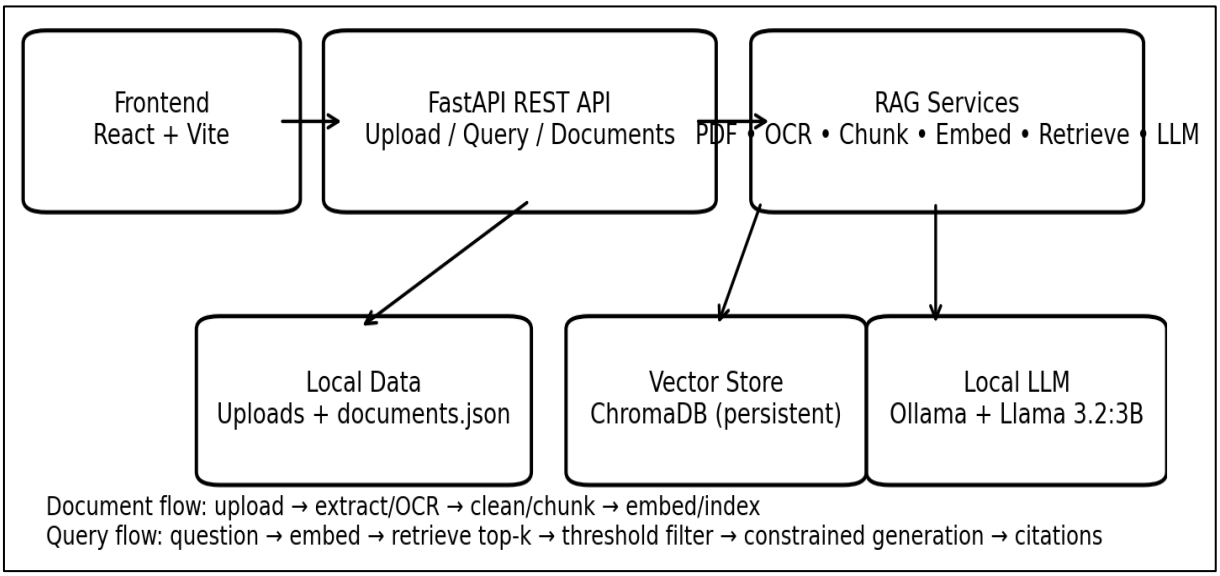


Fig 1 Implemented Multi-PDF RAG Architecture and Data Flow.

Table 1 System Architecture and Components

Layer	Selected technology	Role in the project
Frontend	React + Vite	PDF upload, document list, query input, answers and source display
API	FastAPI + Pydantic	REST endpoints, validation and service orchestration
PDF processing	PyMuPDF + Tesseract OCR	Text extraction with scanned-page fallback
Chunking	Custom paragraph-aware splitter	~700-character chunks with ~100-character overlap
Embeddings	BAAI/bge-small-en-v1.5	384-dimensional dense semantic vectors
Vector DB	ChromaDB PersistentClient	Persistent vectors, documents and metadata
LLM	Ollama + Llama 3.2:3B	Local context-grounded answer generation
Configuration	python-dotenv	Environment-based runtime configuration

The architecture deliberately avoids an external database or hosted LLM dependency in the baseline. Chroma collections support embeddings, documents, metadata and metadata filtering, which matches the project requirement to isolate retrieval to a selected document when a document_id is supplied [6].

➢ *FastAPI Backend*

FastAPI provides the service boundary between the browser and the RAG pipeline. The backend exposes health, PDF upload, document listing, document detail, deletion, and query operations. Pydantic models validate query parameters such as top-k and similarity threshold. The API also applies CORS rules for the local React development server. This separation keeps UI behavior independent from document-processing and retrieval implementation details.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26oct1167

➢ *Document Processing Suite*

PDF processing is page-aware. PyMuPDF extracts normal PDF text and preserves the page number needed for citations. PDF extraction can contain reading-order artifacts or return little text for image-based pages, so OCR is used as a fallback for pages below a configurable text-length threshold. Current OCR metadata records extraction mode (text, OCR, or mixed), OCR page count, text page count, and OCR page numbers. PyMuPDF documentation similarly recommends OCR for image-based text and provides page/block/word extraction options for more structured processing [7], [8].

The cleaned page text is divided into paragraph-aware chunks. The current baseline uses a chunk target of approximately 700 characters with 100 characters of overlap. Chunk metadata includes document_id, filename, page_number and chunk_index. This design balances retrieval granularity with enough local context for the generator.

➢ *Vector Storage and Retrieval*

The embedding stage uses BAAI/bge-small-en-v1.5 through Sentence Transformers. Dense embeddings convert both document chunks and user questions into vectors in the same semantic space. The project normalizes embeddings and uses ChromaDB's cosine-distance index. Retrieval initially requests more candidates than the final top-k value, removes candidates whose distance is above the configured threshold, removes duplicate text, sorts by semantic similarity, and returns the highest-scoring unique chunks. Optional metadata filtering restricts retrieval to a specific document.

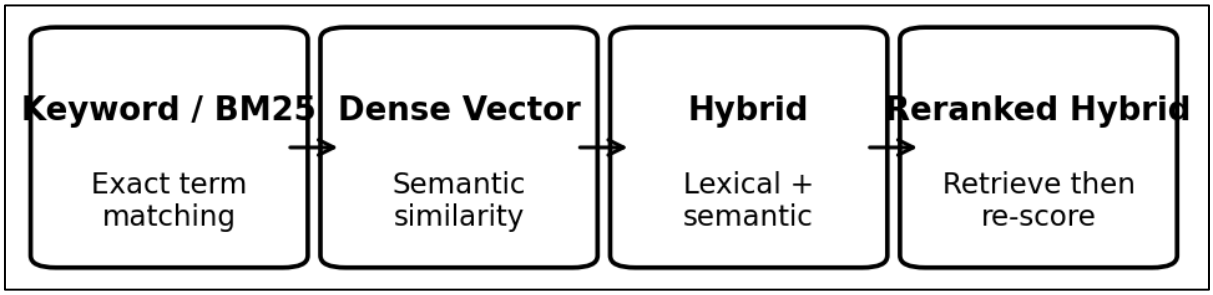


Fig 2 Surveyed Retrieval Progression from Lexical Search to Reranked Hybrid Retrieval.

III. SYSTEM WORKING PRINCIPLE

During question answering, the user submits a natural-language question. The same embedding model converts the question into a vector. ChromaDB retrieves candidate chunks from all indexed PDFs or from a selected document. The retriever applies a similarity threshold, removes duplicate text, sorts candidates, and keeps the requested top-k context. The RAG pipeline formats each source with document, page, chunk and relevance metadata before invoking Llama 3.2:3B.

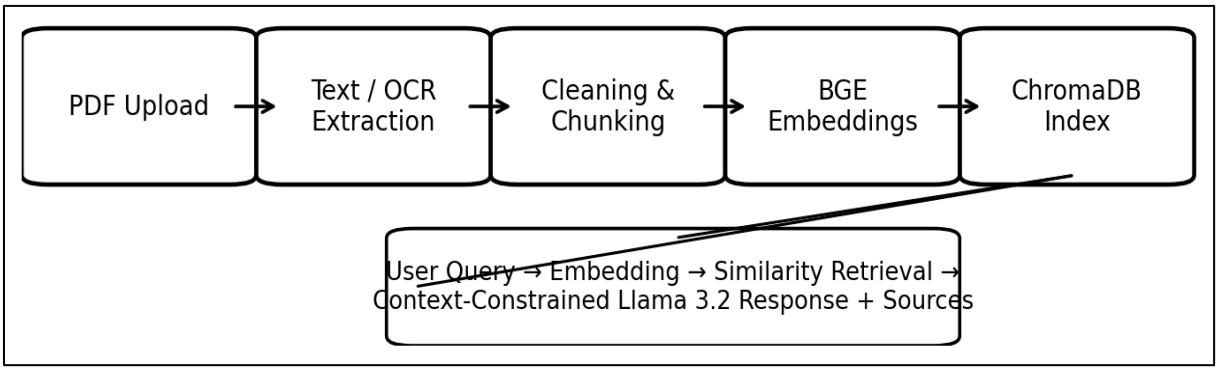


Fig 3 End-to-End Indexing and Question-Answering Lifecycle.

➢ *Similarity-Based Retrieval*

Let q denote the normalized query embedding and d_i a normalized document-chunk embedding. The baseline uses cosine similarity conceptually expressed as:

$$Similarity(q,d_i) = q \cdot d_i$$

Because ChromaDB returns cosine distance, the implementation derives a similarity score as 1 − distance and rejects candidates above the configured distance threshold. The default top-k is 5 and the similarity threshold is 0.50, with the API allowing controlled adjustment. Dense retrieval is supported by the broader literature on semantic passage search [4], while later systems increasingly combine dense and lexical signals when exact terminology is important.

➢ *Controlled Query Processing & Response Generation*

The query endpoint validates the question, confirms that documents exist, optionally verifies the requested document_id, and invokes the RAG pipeline. The LLM receives only the retrieved document context. The generation prompt explicitly states that the supplied context is the only source of truth, prohibits unsupported facts, and requires the exact response “I could not find the answer in the uploaded

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26oct1167

documents." when evidence is insufficient. This is a practical grounding mechanism rather than a guarantee against hallucination.

The response contains the answer together with source citations. A citation identifies the filename, page number, chunk index, similarity and distance. This is important because provenance is one of the motivations for retrieval-augmented generation: external evidence can be inspected rather than treating the model output as an opaque prediction [1].

The current implementation uses a local Llama 3.2:3B model through Ollama with temperature set to zero and a 4096-token context. The local model choice is aligned with the available RTX 3050 4 GB GPU and 16 GB system memory, keeping the baseline practical for development without changing the application architecture.

IV. LITERATURE REVIEW

Lewis et al. introduced RAG as a combination of parametric model memory and non-parametric external memory, demonstrating the value of retrieval for knowledge-intensive generation [1]. Subsequent surveys organize RAG into pre-retrieval, retrieval, post-retrieval and generation stages, emphasizing that performance depends on the interaction of all four stages [2], [3]. This framing maps closely to the present project: PDF extraction and chunking form pre-retrieval processing, ChromaDB performs retrieval, relevance filtering is post-retrieval selection, and Llama 3.2 performs generation.

Dense Passage Retrieval showed that a dual-encoder representation can provide strong semantic passage retrieval compared with traditional sparse retrieval baselines [4]. Sentence-BERT similarly demonstrated that sentence-level embeddings can make semantic similarity search computationally practical [5]. These findings motivate the project's use of normalized dense embeddings rather than keyword-only search.

Document processing remains a critical issue because PDF text is not always represented in natural reading order and scanned pages may contain no machine-readable text. PyMuPDF provides multiple extraction modes and OCR integration, while modern document-conversion systems such as Docling explore layout-aware extraction and table structure recognition [7], [9]. The current project intentionally uses a simpler PyMuPDF + Tesseract path to keep the baseline lightweight.

Long-context research also shows that simply giving an LLM more text does not guarantee reliable use of all evidence. Liu et al. found substantial sensitivity to the position of relevant information within long contexts [10]. This supports the project's retrieval-first design: instead of placing an entire large PDF into the prompt, the system attempts to select a small, relevant evidence set.

RAG evaluation is multi-dimensional. RAGAS proposed reference-free metrics that separate retrieval quality, answer faithfulness and answer relevance [11]. For the present system, evaluation should therefore measure retrieval recall/precision at k, citation correctness, faithfulness, unsupported-answer rate, latency, OCR quality and behavior under multiple PDFs rather than relying only on subjective answer quality.

Recent work also explores GraphRAG for questions requiring global understanding of an entire corpus rather than local passage retrieval [12]. Such methods are relevant to future research but are intentionally outside the current baseline because the project is designed around local passage retrieval and page-level citations.

V. COMPARATIVE ANALYSIS

Table 2 compares common document-QA approaches at the architectural level. The values describe design characteristics rather than measured benchmark scores from this project. No accuracy number is claimed because a formal evaluation dataset has not yet been established.

Table 2 Comparative Analysis of Document-QA Approaches

Approach	Evidence source	Strength	Limitation
Full-document prompting	Entire PDF/context	Simple pipeline	Context cost; irrelevant text; long-context sensitivity
Keyword/BM25	Term statistics	Strong exact-term matching	Can miss semantic paraphrases
Dense vector RAG	Embedding similarity	Semantic matching; simple scaling	Can miss exact identifiers and rare terms
Hybrid retrieval	Sparse + dense	Balances lexical and semantic matching	More components and tuning
Reranked RAG	Initial retrieval + reranker	Improves ordering of candidates	Adds compute and latency
GraphRAG	Graph + summaries	Strong for global corpus questions	Higher indexing complexity
Implemented project	Dense RAG + metadata + threshold	Local, traceable, multi-PDF and OCR-aware	No hybrid/reranking yet

The implemented system occupies the practical baseline point in this design space. It is more robust than a pure full-document prompt because retrieval narrows the context, and it provides more traceability than an unconstrained chatbot. However, hybrid retrieval and reranking are likely to improve difficult cases involving exact names, identifiers, or terminology that a dense model does not rank highly.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26oct1167

➢ *System Implementation*

The prototype is implemented as a modular Python backend and React frontend. The backend is organized into API routes, schemas, configuration, document services, PDF processing, text cleaning, chunking, embeddings, vector storage, retrieval, LLM generation, and the RAG pipeline.

The frontend contains upload, file-list, query, answer, citation and loading components. This modular structure permits individual services to be tested without redesigning the overall system.

The upload path has been tested with multiple PDFs. The project registry records file size, page count, character count, chunk count, vector count, extraction mode and OCR statistics. The current development dataset has included both a one-page resume PDF and a 101-page 10-K PDF, demonstrating that the same collection can index documents of substantially different sizes. OCR support has also been verified on scanned content.

The query path has been tested for normal questions, unsupported questions, and document-specific filtering. The UI displays returned sources so that users can inspect where the answer originated. These tests demonstrate functional behavior, but they should not be interpreted as a formal benchmark; a controlled evaluation set is required for quantitative claims.

➢ *Advantages of the Rag System*

- Grounded answers: generation is constrained to retrieved PDF context.
- Multi-PDF support: several documents can be indexed and searched together.
- Page-level traceability: source metadata preserves filename, page and chunk information.
- OCR support: scanned/image-based PDF pages can enter the same retrieval pipeline.
- Local privacy-oriented deployment: the baseline can run without sending document text to a hosted LLM.
- Modularity: PDF processing, embeddings, retrieval and generation can be evaluated independently.
- Resource awareness: the selected embedding model and 3B local LLM are compatible with the available development hardware.
- Controlled unsupported-question behavior: a not-found response is returned when retrieval provides no acceptable evidence.

➢ *System Limitations*

The current implementation has several practical limitations that affect retrieval quality, document fidelity, and evaluation. Table 3 summarizes the identified limitations, their causes, and the corresponding next steps.

Table 3 System Limitations and Mitigation Directions

Identified limitation	Root cause	Mitigation / next step
OCR errors	Low-quality scans, unusual fonts	Preprocessing, OCR confidence checks, layout-aware OCR
Dense retrieval misses terms	Semantic model may underweight exact identifiers	Hybrid BM25 + dense retrieval
Wrong chunk boundaries	Fixed character target	Semantic/section-aware chunking
Duplicate/near-duplicate context	Overlapping chunks and repeated PDF text	MMR/diversity filtering
Long-context sensitivity	Generator may underuse middle evidence	Context compression and reranking
Tables and layouts	Plain text extraction loses structure	Layout-aware document parsing
No formal benchmark yet	No curated QA test set	RAG evaluation dataset and metrics
Re-indexing required after pipeline changes	Stored vectors reflect old chunks	Document re-index endpoint and versioning

These limitations are consistent with the broader RAG literature. Retrieval quality, context selection, generation faithfulness and document representation all affect the final answer. In particular, long-context behavior and PDF layout complexity mean that simply increasing top-k is not a reliable universal solution [10].

➢ *Practical Applications*

The system is applicable wherever users need question answering over private collections of PDF documents. Examples include academic literature review, research-paper exploration, company reports, technical manuals, policy documents, compliance material, project documentation, resumes and educational notes.

- Academic research: ask questions across multiple research papers and inspect page-level sources.
- Financial and annual reports: retrieve figures, statements and narrative explanations from large filings.
- Enterprise knowledge: search internal manuals, specifications and policy PDFs.
- Education: create a document-grounded study assistant from course material.
- Technical support: retrieve procedures and configuration details from product manuals.
- Document review: compare facts across multiple uploaded reports without manually searching every file.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26oct1167

VI. FUTURE SCOPE & ROADMAP

Future work should improve retrieval and evaluation without replacing the current architecture. The roadmap therefore extends existing services rather than introducing a different application design.

Table 4 Future Research Roadmap

Research domain	Planned enhancement	Target outcome
Re-indexing	Reprocess stored PDFs after OCR/chunking changes	Consistent current vectors and metadata
Evaluation	RAGAS-style faithfulness/relevance plus retrieval metrics	Quantitative system assessment
Hybrid retrieval	BM25 + dense vector retrieval	Better lexical + semantic coverage
Reranking	Cross-encoder or lightweight reranker	Higher precision in top-k context
Query rewriting	Generate retrieval-focused query variants	Improved recall for ambiguous questions
MMR / diversity	Diversify retrieved chunks	Reduce redundant context
Layout-aware parsing	Tables, headings, columns and reading order	Better PDF fidelity
Context compression	Select only answer-bearing spans	Lower prompt noise and latency
Global QA	GraphRAG-style corpus summarization	Better whole-collection questions

A staged implementation is preferable: first add re-indexing and evaluation, then hybrid retrieval and reranking, followed by query rewriting and context compression. These additions preserve the React/FastAPI/PDF-processing/vector-store/LLM architecture while improving individual pipeline stages.

- *Economic Feasibility Analysis*

The baseline has a favorable development cost profile because the principal software components are open-source or locally runnable. React, FastAPI, PyMuPDF, Tesseract, Sentence Transformers, ChromaDB, LangChain and Ollama can be used without a per-query hosted inference charge in the local configuration. The primary hardware requirement is a development machine with sufficient RAM, disk space and GPU/CPU capacity for embedding and local LLM inference.

Table 5 Economic Feasibility of the Local Rag Baseline

Cost factor	Current project approach	Economic implication
LLM inference	Local Ollama + Llama 3.2:3B	No API token fee; uses local compute
Embeddings	Local BGE model	No per-document embedding API fee
Vector storage	Persistent local ChromaDB	Low infrastructure cost for prototype scale
PDF/OCR	PyMuPDF + Tesseract	No hosted OCR dependency in baseline
Frontend/backend	React + FastAPI	Open-source development stack
Scaling	Single-machine prototype	Additional hardware/cloud cost may appear at larger scale

The economic trade-off is therefore between infrastructure simplicity and local compute usage. For a student or research prototype, local execution is attractive because it reduces recurring service costs and keeps document data under the developer's control. At production scale, storage, concurrent inference, GPU capacity, backup, authentication and monitoring become additional cost factors.

VII. CONCLUSION

This survey examined retrieval-augmented generation for multi-PDF question answering and mapped the literature to the implemented RAG System project. The selected architecture combines React and Vite for the interface, FastAPI for the service layer, PyMuPDF with Tesseract OCR for document extraction, paragraph-aware chunking for retrieval units, BAAI/bge-small-en-v1.5 for dense embeddings, persistent ChromaDB for vector storage, and Ollama-hosted Llama 3.2:3B for local grounded generation. The architecture remains unchanged throughout the survey.

The implemented baseline already addresses several practical requirements: multiple PDFs can be indexed in one collection, document metadata can restrict retrieval, scanned pages can be handled through OCR, retrieved chunks are filtered by similarity, and answers expose source information. The main research gap is not basic functionality but systematic evaluation and retrieval quality. Future improvements such as hybrid retrieval, reranking, query rewriting, MMR, context compression, layout-aware extraction and global-corpus methods can be added as enhancements to the same architecture.

Overall, the project represents a practical and extensible document-grounded QA baseline. Its local-first design is suitable for experimentation, academic projects and privacy-sensitive document collections, while its modular services provide a clear path toward a more rigorously evaluated advanced RAG system.

REFERENCES

[1]. P. Lewis et al., “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” arXiv:2005.11401, 2020.

[2]. Y. Gao et al., “Retrieval-Augmented Generation for Large Language Models: A Survey,” arXiv:2312.10997, 2023, revised 2024.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26oct1167

[3]. Y. Huang and J. Huang, “A Survey on Retrieval-Augmented Text Generation for Large Language Models,” arXiv:2404.10981, 2024.
[4]. V. Karpukhin et al., “Dense Passage Retrieval for Open-Domain Question Answering,” Proc. EMNLP, pp. 6769–6781, 2020.
[5]. N. Reimers and I. Gurevych, “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,” Proc. EMNLP-IJCNLP, pp. 3982–3992, 2019.
[6]. Chroma, “Chroma Core Concepts and APIs: Collections, Embeddings, Metadata and Filters,” Chroma Cookbook, accessed 2026.
[7]. PyMuPDF, “The Basics: PDF Text Extraction and OCR,” PyMuPDF Documentation, accessed 2026.
[8]. PyMuPDF, “Text Extraction Recipes and Structured Page Extraction,” PyMuPDF Documentation, accessed 2026.
[9]. C. Auer et al., “Docling Technical Report,” arXiv, 2024.
[10]. N. F. Liu et al., “Lost in the Middle: How Language Models Use Long Contexts,” arXiv:2307.03172, 2023.
[11]. S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, “RAGAS: Automated Evaluation of Retrieval Augmented Generation,” arXiv:2309.15217, 2023.
[12]. D. Edge et al., “From Local to Global: A Graph RAG Approach to Query-Focused Summarization,” arXiv:2404.16130, 2024.
[13]. BAAI, “BGE Embedding Models,” BAAI FlagEmbedding project and model documentation, accessed 2026.
[14]. Ollama, “Llama 3.2 Model and Local Inference Documentation,” Ollama Documentation, accessed 2026.
[15]. LangChain, “Retrieval and RAG Concepts,” LangChain Documentation, accessed 2026.
[16]. F. M. Reimers and I. Gurevych, “Making Monolingual Sentence Embeddings Multilingual Using Knowledge Distillation,” EMNLP, 2020.