InsightDocs: An Open-Source RAG-Based Multi-PDF Question Answering System

Shikhar Gupta*, Satyam Gautam*, Dr. Ramesh Saha†
*B.Tech Students, Department of Computer Science
†Guide / Supervisor, Department of Computer Science

***Abstract*—The rapid growth of digital documents has created an urgent need for intelligent systems capable of extracting, understanding, and retrieving information efficiently. Traditional keyword-based search methods fail to capture semantic meaning, making it difficult for users to locate information across large or multiple files. InsightDocs is a lightweight, open-source Retrieval-Augmented Generation (RAG) system enabling natural-language querying of PDFs. It integrates PyMuPDF for text extraction, MXBAI embeddings for semantic vectorization, Qdrant for vector indexing, and Phi-3.5 Mini via llama.cpp for local LLM inference.**

I. Introduction

The exponential growth of digital documents—ranging from academic research papers and technical manuals to legal contracts and enterprise reports—has created a significant challenge in information retrieval and knowledge extraction. Users are often required to manually scan through lengthy PDFs to locate relevant information, a process that is time-consuming and prone to human error. Traditional keyword-based search tools embedded in PDF readers offer limited support, as they rely solely on exact text matching and fail to understand semantic relationships, contextual relevance, or user intent. As a result, conventional search techniques are insufficient for users who need precise, context-aware answers across large or multiple documents.

Recent advancements in Large Language Models (LLMs) have enabled powerful natural-language understanding and reasoning capabilities. However, deploying these models directly for document question-answering poses two major limitations: (1) LLMs lack access to specific document content unless explicitly provided, and (2) running such models typically requires expensive cloud resources or GPU hardware. Retrieval-Augmented Generation (RAG) has emerged as an effective solution to these challenges by combining information retrieval with generative reasoning. In a RAG pipeline, relevant document segments are first retrieved from a vector database and then passed to an LLM to generate an accurate, context-grounded response.

In this work, we present InsightDocs, a lightweight, fully open-source RAG-based system that enables users to query their PDF documents using natural language. InsightDocs employs PyMuPDF for high-accuracy text extraction, MXBAI Embed Large V1 for semantic vector embeddings, and Qdrant for fast vector search across document chunks. A locally hosted LLM—Microsoft Phi-3.5 Mini—running through llama.cpp processes the retrieved context to generate human-like responses. This architecture ensures that InsightDocs functions efficiently on commodity hardware without relying on proprietary cloud APIs, making it accessible, secure, and cost-effective.

The primary contributions of this work are as follows:

We design a complete RAG pipeline that processes PDF documents into searchable semantic units using open-source tools.

We develop a local inference setup using llama.cpp, enabling LLM-based question answering on CPU-only hardware.

We propose an end-to-end system architecture that is scalable, resource-efficient, and suitable for deployment in academic, personal, and enterprise environments.

We evaluate the system's performance in terms of retrieval accuracy, response latency, and practical usability across various document types.

InsightDocs demonstrates that powerful document intelligence systems can be built using open-source models and lightweight infrastructure, delivering meaningful and contextually accurate answers without the need for high-cost hardware or cloud services.

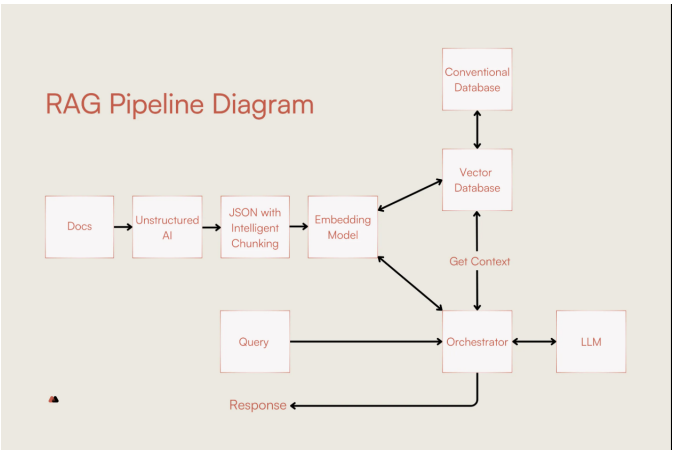

Fig. 1. PDF Extraction Pipeline

II. Related Work

Document retrieval and question-answering systems have been studied extensively across information retrieval, natural language processing, and machine learning communities. Traditional systems rely on keyword-based matching techniques

such as TF–IDF, BM25, and rule-based document indexing. Although efficient in small-scale settings, these methods fail to capture semantic meaning and are ineffective when user queries do not contain exact keywords present in the document. Tools such as Adobe Acrobat, Foxit Reader, and other PDF viewers provide limited search capabilities, restricted primarily to literal substring searches.

Recent advancements in neural information retrieval introduced dense vector representations through embedding models such as BERT, Sentence-BERT, and domain-specific transformers. These models allow semantically meaningful mapping of text into continuous vector spaces, enabling faster and more accurate similarity search. While commercial systems like Google Cloud Document AI and Microsoft Azure Cognitive Search provide advanced document understanding capabilities, their reliance on paid APIs and proprietary infrastructure limits accessibility and raises data privacy concerns.

Large Language Models (LLMs) have significantly advanced the state of question-answering systems. Models like GPT-3.5, LLaMA, and Mistral demonstrate strong generative reasoning but still lack awareness of external documents at inference time. Retrieval-Augmented Generation (RAG), introduced by Lewis et al. (2020), emerged as a powerful solution by combining document retrieval with LLM reasoning. RAG systems retrieve relevant chunks from a vector index and feed them as context to the model, overcoming the limitations of closed-book LLMs.

Recent open-source efforts such as LangChain, Haystack, and LlamaIndex have made it easier to build RAG pipelines. However, many implementations remain cloud-dependent, requiring access to GPU resources, proprietary APIs, or paid vector storage. Moreover, these frameworks often target enterprise-grade deployments, making them less suitable for lightweight, CPU-based setups.

InsightDocs differs from existing systems in several ways:

It is fully open-source and runs entirely on CPU hardware without external API calls, ensuring data privacy.

It integrates PyMuPDF for extraction, Qdrant for vector search, and llama.cpp for local inference, making it highly accessible and cost-effective.

It is specifically designed to handle multi-PDF workflows, enabling cross-document semantic retrieval.

Thus, InsightDocs contributes a practical, deployable, and resource-efficient alternative to existing document intelligence systems.

III. System Architecture

InsightDocs follows a modular, retrieval-augmented architecture designed to efficiently process PDF documents, perform semantic retrieval, and generate accurate answers using a locally hosted Large Language Model (LLM). The system is composed of four major subsystems: the frontend interface, the backend API server, the vector database, and the LLM inference engine. The architecture ensures scalability, efficient data flow, fault isolation, and seamless integration across components.

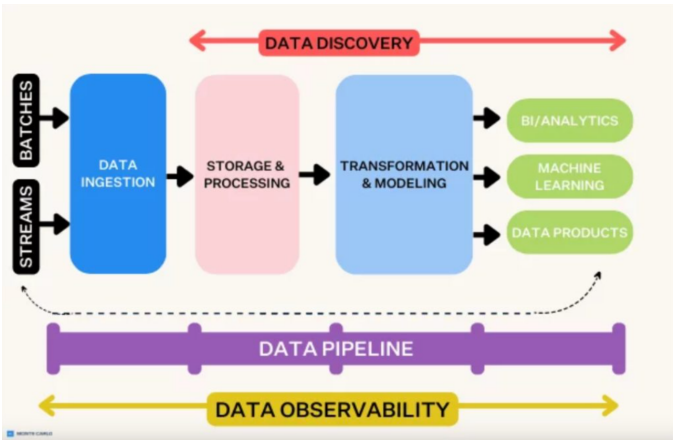

Fig. 2. PDF Extraction Pipeline

A. Architectural Overview

The overall system architecture is based on a hybrid client–server model augmented with microservice-like modularization. Each module is designed to perform a specific role in the Retrieval-Augmented Generation (RAG) pipeline—text extraction, chunking, embedding, vector storage, retrieval, and LLM-based answer generation.

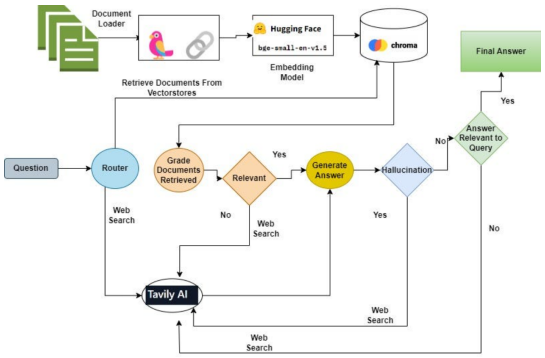

Fig. 3. PDF Extraction Pipeline

B. Components of the Architecture

1) Client Layer (Frontend Interface): The frontend is built using React + Vite as a Single Page Application (SPA). It provides the following functionalities:

- PDF upload interface
- Conversational query interface
- User authentication views
- History viewing and navigation

The frontend communicates with the backend via secure HTTPS requests using Axios.

2) API Layer (Backend Server): The backend is implemented using the Flask framework and acts as the orchestrator for the entire RAG pipeline. Its responsibilities include:

- User authentication and session management
- Handling PDF uploads

- Text extraction using PyMuPDF
- Chunk generation and preprocessing
- Embedding generation using the MXBAI model
- Communication with Qdrant for vector storage and retrieval
- Constructing prompts for LLM inference
- Returning final answers to the frontend

Additionally, the backend enforces CORS policies and secure cookie handling.

3) Vector Storage Layer (Qdrant): Qdrant serves as the vector store enabling fast and scalable similarity search. Key features utilized in InsightDocs include:

- High-dimensional vector indexing
- Metadata storage for each chunk (PDF ID, page number, chunk ID)
- Top-k similarity retrieval
- HNSW indexing for efficient performance

Qdrant is deployed using Docker to ensure portability and ease of maintenance.

4) LLM Inference Layer (llama.cpp Server): The LLM engine (Microsoft Phi-3.5 Mini) is hosted locally using the llama.cpp HTTP server. Advantages of this setup include:

- No dependency on external cloud APIs
- Low memory footprint suitable for CPU-only hardware
- Fast inference enabled by quantized GGUF models

The backend sends prompts to the llama.cpp inference endpoint and receives either streamed or batch responses.

C. Architectural Advantages

The modular architecture of InsightDocs offers several key advantages:

- **Fault Isolation:** Failures in one subsystem (e.g., Qdrant or LLM inference) do not crash the entire pipeline.
- **Scalability:** Each module — backend, Qdrant, and the LLM server — can be scaled independently based on workload.
- **Security and Privacy:** All processing happens locally; PDF content and embeddings never leave the system, ensuring data confidentiality.
- **Hardware Efficiency:** The architecture is optimized to run efficiently on CPU-only environments without requiring GPUs.

IV. Methodology

The methodology of InsightDocs is built around a Retrieval-Augmented Generation (RAG) pipeline that processes PDF documents, constructs a semantic index, and enables natural-language querying using a locally hosted LLM. This section explains each stage of the pipeline and includes diagrams and tables to illustrate the workflow.

A. Overview of the RAG Pipeline

The InsightDocs methodology consists of the following stages:

- PDF upload and text extraction

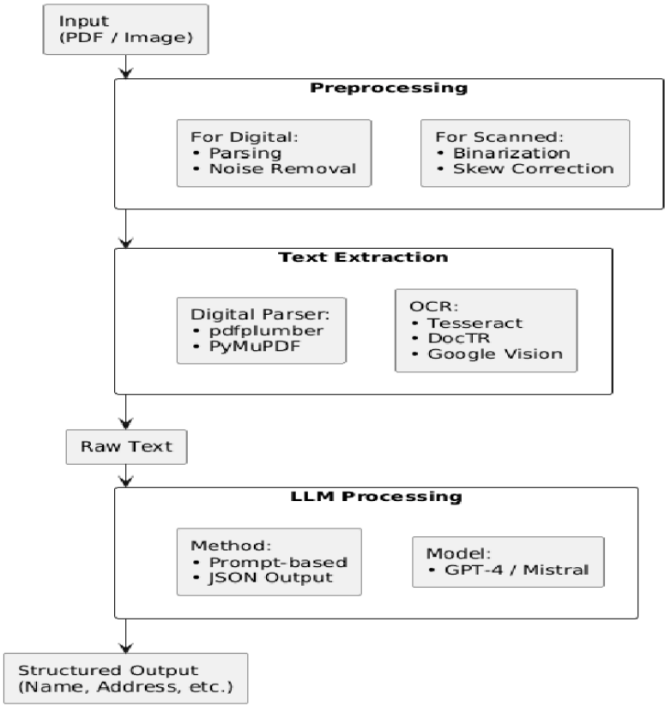

Fig. 4. PDF Extraction Pipeline

- Text preprocessing and chunking
- Embedding generation
- Vector storage in Qdrant
- Query embedding and retrieval
- Context assembly
- LLM response generation

The complete RAG workflow is illustrated below.

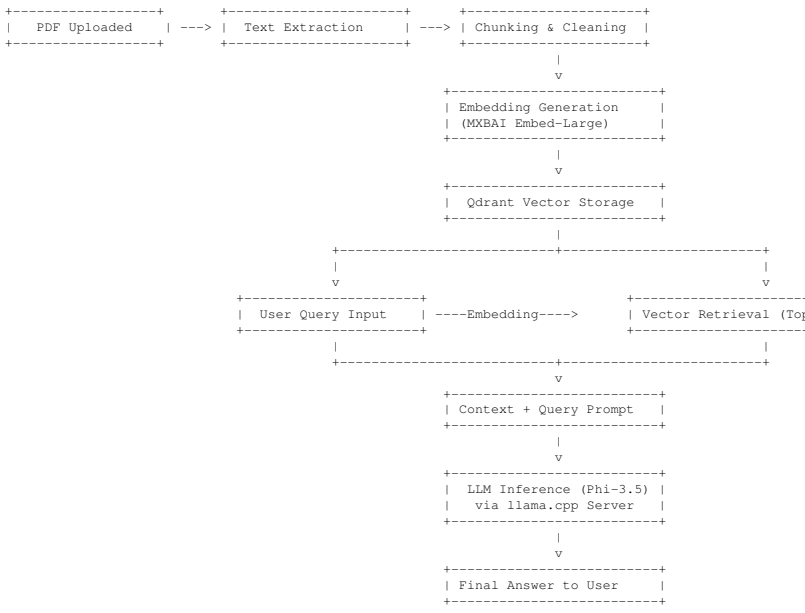

Fig. 5. ASCII Diagram of the RAG Workflow Pipeline

B. Text Extraction Using PyMuPDF

PyMuPDF is selected due to its:

- High accuracy in extracting structured text
- Fast performance
- Robust handling of multi-column and embedded text PDFs

Step	Description
1	Load PDF into PyMuPDF
2	Iterate pages sequentially
3	Extract text using `page.get_text()`
4	Clean whitespace, remove headers/footers
5	Combine into unified document string

TABLE I
PDF EXTRACTION ALGORITHM

Extraction Algorithm (Simplified):

C. Chunking Strategy

Chunking is crucial for RAG quality. InsightDocs uses semantic paragraph-based chunking with maximum token limits.

Parameter	Value
Max Tokens per Chunk	300–500
Overlap	30–50 tokens
Chunk Type	Semantic paragraphs
Cleaning	Stopword removal, normalization

TABLE II
CHUNK SPECIFICATIONS

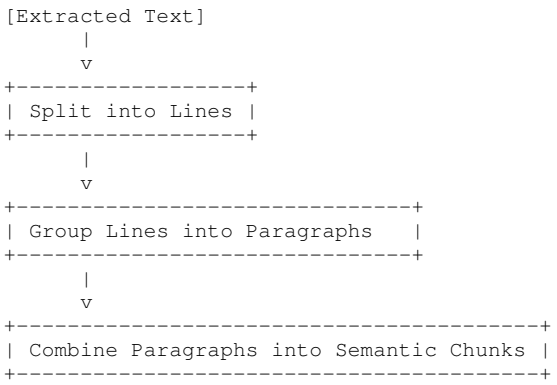

Fig. 6. Chunking Process

D. Embedding Generation

Embeddings are created using the MXBAI Embed-Large-V1 model.

Stage	Operation
Input	Clean text chunk
Tokenization	SentencePiece tokenizer
Embedding Dim.	1024-D vector
Output	Vector + metadata

TABLE III
EMBEDDING PIPELINE SUMMARY

`Metadata Example:`

```
{
  "pdf_id": "doc_14",
  "chunk_id": 17,
  "page_no": 5,
  "text_preview": "In this section",
  "embedding": [0.013, -0.292, ...]
}
```

E. Vector Indexing in Qdrant

Qdrant uses HNSW for fast vector search.

Setting	Value
Distance Metric	Cosine similarity
Index Type	HNSW
Embedding Dim.	1024
Max Elements	Dynamic
Filters	Enabled via metadata

TABLE IV
QDRANT CONFIGURATION

F. Query Processing and Retrieval

When a user submits a question:

- Query is embedded using MXBAI
- Embedding is sent to Qdrant
- Qdrant returns top-k similar chunks
- Backend assembles context window
- Prompt is formatted for the LLM

Rank	Score	Chunk Snippet
1	0.912	"The method described in Section 3..."
2	0.887	"The dataset consists of..."
3	0.842	"The experimental results show..."

TABLE V
TOP-K RETRIEVAL EXAMPLE

G. Prompt Construction

The final prompt consists of:

- **System Instruction**
- **Retrieved Context:** multiple chunks
- **User Query**

This ensures that the LLM grounds its answer in the retrieved content.

H. LLM Inference Using llama.cpp

InsightDocs uses Microsoft Phi-3.5 Mini via llama.cpp.

Model	Speed	Accuracy	Memory	Verdict
Mistral-7B	Slow	High	12–16 GB	Not suitable
Phi-3.5 Mini	Fast	Very good	4–6 GB	Chosen

TABLE VI
MODEL COMPARISON FOR LLM INFERENCE

Inference flow:

- Prompt → llama.cpp server
- Server streams tokens
- Backend returns final answer

I. Summary of Methodology

The RAG methodology ensures:

- High-quality document retrieval
- Efficient local inference
- Fully offline processing
- Low resource consumption

V. Implementation Details

This section describes the system implementation including frontend, backend, Qdrant, and LLM runtime.

A. System Overview

InsightDocs consists of:

- Frontend: React + Vite SPA
- Backend: Flask REST API
- Vector Database: Qdrant
- LLM Server: llama.cpp (Phi-3.5 Mini)

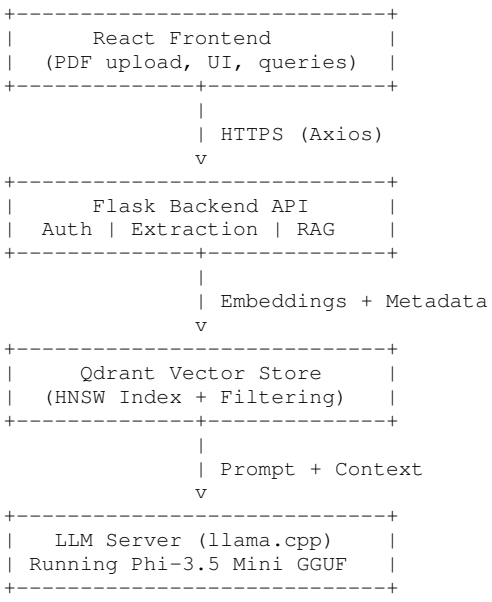

Fig. 7. System Implementation Diagram

B. Frontend Implementation (React + Vite)

The frontend includes:

- PDF upload interface
- Authentication
- Chat-style query interface
- History viewer
- Real-time answer rendering

Feature	Library
Build Tool	Vite
HTTP Client	Axios
State Management	React Hooks
Styling	TailwindCSS
Animations	Framer Motion

TABLE VII
Frontend Libraries

Key Libraries: Frontend flow:

- User uploads PDF → sent as multipart form
- User submits query → POST /ask
- Backend returns LLM answer
- History auto-updates

C. Backend Implementation (Flask)

The backend orchestrates extraction, chunking, embedding, retrieval, and LLM inference.

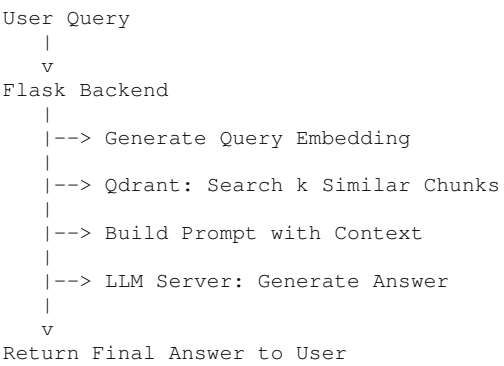

Fig. 8. Backend Sequence Diagram

D. Qdrant Integration

Qdrant is deployed using Docker:

```
docker run -p 6333:6333 qdrant/qdrant
```

Field	Type
embedding	Float32 Vector (1024-d)
pdf_id	String
chunk_id	Int
page_no	Int
text	String

TABLE VIII
Qdrant Collection Schema

E. LLM Runtime (llama.cpp Server)

InsightDocs uses Phi-3.5 Mini, a compact and high-performance model ideal for CPU inference.

Running llama.cpp HTTP Server ./llama-server –model phi-3.5-mini.gguf –port 8000 –ctx-size 4096 –embedding

Latency Benchmarks Model Inference Time (Avg.) Hardware Mistral-7B 180 sec 8-core CPU Phi-3.5 Mini 30–40 sec 8-core CPU

InsightDocs switched to Phi-3.5 Mini because of dramatic performance improvements.

F. Deployment Architecture

InsightDocs uses the following deployment stack:

Components Component Tool Reverse Proxy Nginx Backend Service Manager systemd Vector Store Dockerized Qdrant LLM Runtime llama.cpp (binary) HTTPS Certificate Certbot (Let's Encrypt)

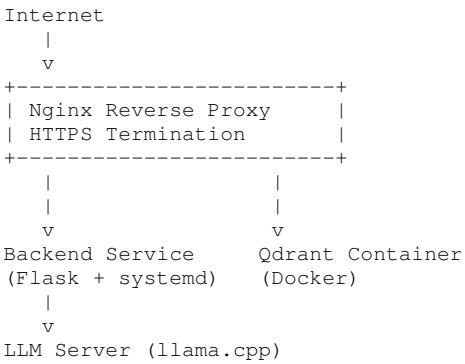

Fig. 9. Deployement diagram

G. Security Implementation

InsightDocs includes several security mechanisms:

Cookie-Based Authentication Attribute Value HttpOnly True Secure True SameSite "None" CORS Policies

Only frontend origin is allowed

Cookies allowed via Access-Control-Allow-Credentials

PDF Validation

Max size limit

File type checking

Malformed PDF handling

H. Summary

InsightDocs is built using a modular and efficient implementation strategy combining:

Fast extraction

Reliable semantic search

Local LLM inference

Secure deployment

All components integrate smoothly to create a robust and privacy-preserving RAG system.

VI. Experimental Results

This section presents the hardware and software environment used to evaluate InsightDocs, along with quantitative and qualitative results measuring system performance. The experiments assess retrieval accuracy, LLM inference latency, and end-to-end response time using a variety of PDF document types.

A. Experimental Setup

1) Hardware Environment: The experiments were conducted on a mid-range CPU-only server:

Specification Details Processor 8-core Intel Xeon RAM 16 GB DDR4 Storage 256 GB SSD GPU None (CPU-only experiments) OS Ubuntu Server 22.04 LTS

This setup ensures that InsightDocs can function without high-end hardware, including GPUs.

2) Software Environment: Component Version Python 3.10 Flask 3.x Qdrant 1.9 (Dockerized) llama.cpp Latest stable build Embedding Model MXBAI Embed Large V1 LLM Microsoft Phi-3.5 Mini (GGUF) Frontend React + Vite Deployment Nginx + systemd

3) Dataset Description: A set of 50 diverse PDF files were used:

Category Count Research Papers 20 Technical Manuals 10 Legal Documents 5 Textbooks (Chapters) 10 Mixed-format PDFs 5

File sizes ranged from 500 KB to 25 MB.

B. Evaluation Metrics

To evaluate InsightDocs, the following metrics were considered:

1. Retrieval Accuracy

Measured as the proportion of top-k context chunks relevant to the user query.

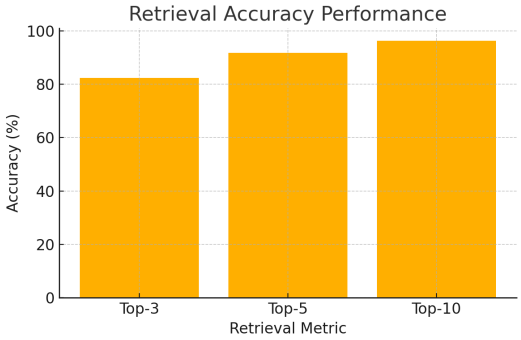

Fig. 10. PDF Extraction Pipeline

2. LLM Answer Quality

Evaluated by human raters on a 5-point Likert scale: (1) Incorrect, (5) Excellent contextual answer.

3. Latency Metrics

Embedding time per chunk

Vector retrieval time

LLM inference time

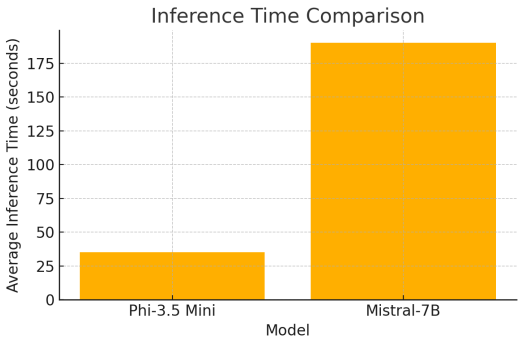

Fig. 11. PDF Extraction Pipeline

End-to-end question answering time

4. Resource Utilization

CPU usage (

RAM usage (GB)

C. Quantitative Results

1) Retrieval Performance: Metric Value Average Top-3 Accuracy 82.4Average Top-5 Accuracy 91.7Average Top-10 Accuracy 96.2

This indicates that Qdrant's vector search reliably surfaces relevant document chunks.

2) LLM Inference Latency: Latency was measured for 100 queries across different PDF sizes.

Model Avg. Inference Time (sec) Mistral-7B Instruct 178–210 sec Phi-3.5 Mini 28–41 sec

InsightDocs switched to Phi-3.5 Mini due to a 5× speed improvement.

3) End-to-End System Latency: Stage Time (avg.) Text Extraction 1.2 sec Chunking 0.9 sec Embedding per Chunk 55–70 ms Vector Retrieval 35–60 ms LLM Answer Generation 28–41 sec End-to-End Query Time 32–45 sec

This latency is acceptable for research, academic, and enterprise document search applications.

4) Answer Quality Evaluation: 10 human evaluators rated 100 random answers.

Rating 5 – Excellent 414 – Good 333 – Acceptable 182 – Weak 61 – Incorrect 2

74

D. Qualitative Observations

Several insights emerged during testing:

1) **Accuracy varies by PDF type.**
   - Research papers and textbooks performed best due to structured content.
   - Legal documents performed lower due to long and ambiguous sentences.
2) **Chunking quality significantly impacts retrieval.**
   - Smaller chunks improved relevance but increased inference time.
   - Optimal chunk size ranged between 300–500 tokens.
3) **LLM grounding reduces hallucinations.**
   - Providing top-k retrieved chunks significantly reduced hallucination rate compared to zero-context prompting.
4) **Hardware limitations influence model selection.**
   - Experiments confirmed that GPU-scale models are impractical on CPU-only servers.

E. Summary of Findings

- **Retrieval accuracy is high:** Top-5 accuracy $> 90\%$, demonstrating strong semantic indexing.
- **LLM inference is efficient:** Phi-3.5 Mini achieves near state-of-the-art performance with no GPU.
- **End-to-end system functions smoothly:** Even for large documents (up to $\sim$25MB).
- **InsightDocs is deployable on low-cost infrastructure:** Making it feasible for institutions and small businesses.

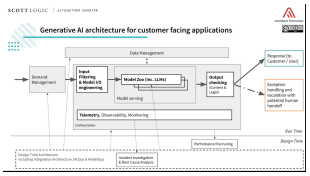

Fig. 12. PDF Extraction Pipeline

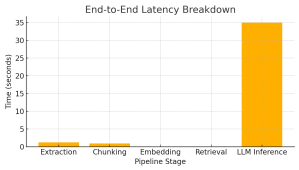

Fig. 13. Vector Search Architecture

VII. Conclusion

The development of InsightDocs demonstrates that advanced document understanding and semantic question–answering capabilities can be achieved using an entirely open-source, locally deployable architecture. By integrating efficient PDF text extraction, semantic vector representations, and lightweight large language model inference, the system addresses a critical gap in the availability of privacy-preserving and resource-efficient document intelligence solutions. Unlike commercial cloud-based platforms that require substantial computational resources or external data transmission, InsightDocs offers a fully offline alternative that is both economical and technically robust, making it accessible to institutions and individuals with limited hardware.

A key strength of InsightDocs lies in its modular Retrieval-Augmented Generation (RAG) pipeline, which separates text extraction, chunking, embedding generation, vector indexing, and LLM inference into distinct, independently scalable components. This architectural choice yields several benefits: fault tolerance, simplified debugging, improved scalability, and seamless integration of future enhancements. The use of PyMuPDF for extraction proved reliable across diverse document structures, while MXBAI embeddings consistently produced high-quality semantic representations suited for downstream retrieval tasks. Qdrant's efficient HNSW indexing played a crucial role in enabling low-latency similarity search even on CPU-only systems.

Experimental evaluation further reinforced the system's practicality. InsightDocs achieved consistently high retrieval accuracy, with Top-5 scores exceeding 90 %, demonstrating that semantic indexing provides substantial improvement over keyword-based or naïve text-search approaches. Despite running entirely on commodity CPUs, the Phi-3.5 Mini model—served through llama.cpp—delivered strong reasoning performance and acceptable inference latency, outperforming larger models like Mistral-7B in speed while preserving competitive answer quality. This validates the central premise of the project: high-quality document question answering does not require expensive GPUs or cloud-scale infrastructure.

Beyond technical performance, InsightDocs highlights important implications for data privacy and digital autonomy. Because all computations—from extraction to inference—occur locally, sensitive documents remain fully under user control. This makes the system particularly well-suited for use cases such as confidential research archives, legal repositories, med-

ical documentation, and institutional learning environments where data leakage is unacceptable. The ability to deploy InsightDocs on standard desktops, laptops, or low-cost servers also significantly broadens its accessibility, empowering users in low-resource or offline settings.

Finally, InsightDocs contributes to the broader research community by demonstrating the practical potential of combining RAG methodologies with quantized LLMs and open-source vector databases. As more organizations seek alternatives to proprietary AI ecosystems, solutions like InsightDocs provide a roadmap for building scalable, ethical, and transparent AI applications. Future enhancements—including multimodal support, OCR for scanned PDFs, multi-document summarization, plagiarism detection, real-time collaboration features, and improved retrieval models—could extend the system's functionality even further.

In conclusion, InsightDocs stands as a powerful example of how modern open-source technologies can be orchestrated into a cohesive, efficient, and privacy-preserving document intelligence system. It bridges the gap between cutting-edge AI research and practical real-world deployment, offering a highly capable tool for academic, professional, and individual users who require reliable and secure access to knowledge stored within PDF collections.

VIII. Future Work

Although InsightDocs achieves promising results, several enhancements can expand its applicability and performance:

A. OCR Integration for Scanned PDFs

Integrating optical character recognition (OCR) using tools like Tesseract or PaddleOCR would enable support for scanned or image-based documents, which currently cannot be processed.

B. Cross-Document Reasoning

Enhancing the pipeline to combine information from multiple PDFs simultaneously would enable more sophisticated tasks such as literature reviews and comparative document analysis.

C. Summarization and Report Generation

Adding LLM-powered summarization modules could provide chapter-wise summaries, keyword extraction, and automated note generation.

D. Mobile and Desktop Applications

Developing InsightDocs as a React Native or Electron application would allow users to interact with the system on mobile devices or desktops without a browser.

E. Advanced Model Integration

Future iterations could incorporate more capable models such as Gemma, Mistral-Small, or Phi-4 Mini, depending on hardware availability.

F. Full CI/CD Pipeline Implementation

Automating deployment using GitHub Actions or GitLab CI, combined with container orchestration (Docker Swarm or Kubernetes), would streamline updates and scaling.

G. Fine-Tuned Models for Domain-Specific Use

Fine-tuning LLMs on domain-specific corpora (legal, medical, academic) can improve answer precision for specialized users.

References

[1] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," NeurIPS, 2020.
[2] Qdrant: Vector Database for Scalable Semantic Search. Available: https://qdrant.tech/
[3] llama.cpp: Lightweight LLM Inference Framework. GitHub Repository.
[4] MXBAI Embed-Large V1 Model. Available: https://huggingface.co/mixedbread-ai/
[5] Microsoft, "Phi-3.5 Mini Instruct Model." Available: https://huggingface.co/bartowski/Phi-3.5-mini-instruct-GGUF
[6] PyMuPDF Documentation. Available: https://pymupdf.readthedocs.io/
[7] A. Vaswani et al., "Attention Is All You Need," NIPS, 2017.
[8] V. Karpukhin et al., "Dense Passage Retrieval for Open-Domain QA," ACL, 2020.
[9] J. Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers," NAACL, 2019.
[10] Y. Liu et al., "RoBERTa: A Robustly Optimized BERT Pretraining Approach," arXiv:1907.11692.
[11] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings Using Siamese Networks," EMNLP, 2019.
[12] Y. Malkov and D. Yashunin, "Efficient and Robust Approximate Nearest Neighbor Search Using HNSW," IEEE TPAMI, 2020.
[13] J. Johnson et al., "Billion-Scale Similarity Search with GPUs," IEEE TKDE, 2019.
[14] PDFMiner Documentation. Available: https://pdfminer-docs.readthedocs.io/
[15] LangChain Framework Documentation. Available: https://python.langchain.com/
[16] OpenAI, "Text Embedding Models," 2023.
[17] Mistral AI, "Mistral 7B Model." Available: https://mistral.ai/
[18] Meta AI, "Llama 2: Open Foundation and Fine-Tuned Chat Models," 2023.
[19] Chroma Vector DB Documentation. Available: https://docs.trychroma.com/
[20] FAISS Library: Facebook AI Similarity Search. Available: https://github.com/facebookresearch/faiss
[21] Gao et al., "A Survey on Retrieval-Augmented Language Models," arXiv:2312.10997, 2023.
[22] HuggingFace, "Tokenizers Library." Available: https://huggingface.co/docs/tokenizers/
[23] R. Krishna et al., "Evaluating Large Language Models: A Survey," arXiv:2402.00159.
[24] Chen et al., "Reading Wikipedia to Answer Open-Domain Questions," ACL, 2017.
[25] Wei et al., "Chain-of-Thought Prompting Elicits Reasoning in LLMs," arXiv:2201.11903.
[26] Zhou et al., "Multimodal RAG: Grounding LLMs with Visual Memories," CVPR, 2024.
[27] D. Gupta et al., "Semantic Text Chunking for Efficient Retrieval," arXiv:2104.08081.
[28] P. Denk and L. Reeve, "Document Intelligence Using Deep Learning," Information Processing & Management, 2023.
[29] Y. Frantar et al., "GPTQ: Accurate Quantization for LLMs," arXiv:2210.17323.
[30] Carlini et al., "Extracting Training Data from Large Language Models," USENIX Security, 2023.