

DocAssistAI: An Intelligent Document Question Answering System Using LLM's and RAG

Mrs. Sreevani¹, Challa Harshitha², Bezawada Urmitha³, Ch. Kiran Kumar⁴, G. Tanay Krishna⁵, Bhanu Prakash⁶.

¹Assistant Professor, Dept. of Computer Science and Engineering, HITAM, Telangana, India

²³⁴⁵⁶Student, Dept. of Computer Science and Engineering, HITAM, Telangana, India

Abstract - DocAssist AI is an intelligent document analysis platform designed to streamline automated information retrieval from complex files, including PDF, DOCX, and TXT formats. Uploaded files and web contents undergo validation, text extraction, and paragraph-aware chunking with overlapping windows to preserve contextual integrity. High-dimensional SentenceTransformer embeddings and a persistent ChromaDB vector store enable fast, document-scoped semantic similarity retrieval. The retrieved context snippets are processed by Google's Gemini Large Language Model through a Retrieval-Augmented Generation (RAG) framework to generate precise, grounded answers with citations. Multi-turn session memory and robust security mechanisms, including JWT authentication, SSRF protection, and strict data isolation, ensure safe and seamless user interactions. By integrating vector search with generative AI through a React-Flask architecture, DocAssist AI provides a scalable, secure, and user-centric solution for document intelligence.

Keywords: Retrieval-Augmented Generation (RAG), Document Intelligence, Large Language Models (LLM), Google Gemini AI, ChromaDB, SentenceTransformers, Vector Embeddings, Semantic Search, Flask REST API, React.js, Multi-Tenant Security, Natural Language Processing (NLP).

1. INTRODUCTION

The exponential growth of digital documents across academic, industrial, and legal domains has made efficient information retrieval a critical necessity. Researchers, students, and professionals frequently interact with lengthy PDF articles, technical specifications, project reports, and scanned documentation. While digital storage makes sharing convenient, locating specific information within voluminous documents remains a significant challenge.

Recent advancements in Natural Language Processing (NLP) and Large Language Models (LLMs) have enabled conversational interfaces capable of understanding complex human queries. However, standard language models operate strictly on static pre-training data and

cannot inherently access private or newly uploaded user documents. To bridge this gap, DocAssist AI introduces an end-to-end, document-grounded question-answering system based on Retrieval-Augmented Generation (RAG), allowing users to upload documents and interact with their content using intuitive natural language questions.

1.1 NEED FOR DOCASSIST AI

Conventional document search mechanisms primarily rely on keyword matching (e.g., standard Ctrl+F functionality or inverted index systems). While keyword search is computationally fast, it exhibits severe limitations when user queries involve natural language variations, semantic synonyms, or conceptual questions where exact vocabulary is not shared between the prompt and the document text. Furthermore, reading through extensive documents to locate specific parameters requires substantial cognitive effort and time. Additionally, standalone LLMs pose two critical limitations when applied to document-based tasks:

1. Lack of Private Knowledge: Base models have no native access to user-uploaded, proprietary, or domain-specific files.
2. Hallucination Risk: When asked about specific document contents without restricted context, base LLMs tend to generate plausible yet unverified or incorrect information.

DocAssist AI is specifically designed to address these limitations. The system integrates multi-format text extraction (PyMuPDF, python-docx, BeautifulSoup), Optical Character Recognition (OCR fallback via Tesseract), paragraph- and sentence-aware chunking, dense vector embeddings (all-MiniLM-L6-v2), document-scoped vector retrieval (ChromaDB), hybrid keyword-dense search, and context-restricted response generation using Google's Gemini LLM. By returning precise, citation-backed answers ([C1], [C2]), DocAssist AI eliminates manual document searching and guarantees factual grounding.

2. LITERATURE REVIEW

Document Information Retrieval (IR) and Question Answering (QA) have evolved through distinct

technological phases:

source chunks

2.1 Keyword-Based Retrieval Systems

Traditional IR systems rely on Lexical Matching models such as Term Frequency-Inverse Document Frequency (TF-IDF) and BM25. These algorithms index documents by token frequencies and retrieve text segments containing matching terms. While effective for exact keyword queries, lexical search fails when synonyms are used, or when queries require semantic comprehension.

2.2 Dense Vector Embeddings & Semantic Search

The introduction of Transformer-based models (Vaswani et al., 2017) and pre-trained language representations like BERT (Devlin et al., 2019) enabled dense semantic vector embeddings. SentenceTransformers (Reimers & Gurevych, 2019) map arbitrary text spans into high-dimensional vector spaces where semantically similar passages reside close to one another based on cosine distance or Euclidean distance. Vector databases such as ChromaDB allow real-time indexing and similarity search over millions of dense vectors.

2.3 Retrieval-Augmented Generation

(RAG) Machine Learning techniques have been widely used to analyze software data and predict defect-prone components. These algorithms identify relationships between historical software metrics and defect occurrence.

Introduced by Lewis et al. (2020), RAG decouples knowledge retrieval from text generation. Instead of fine-tuning an LLM on domain documents, RAG queries an external index, extracts relevant context passages, and injects them into the LLM's prompt window. This architecture ensures that LLM outputs remain strictly grounded in verified facts while enabling dynamic updates to the knowledge base without retraining costs.

2.4 Research Gap Addressed by DocAssist AI

Many existing RAG tools suffer from text-extraction failures when handling scanned or image-heavy PDFs, lack document-level access control, fail to retrieve technical acronyms using vector search alone, or omit explicit source citation markers. DocAssist AI addresses these gaps by combining:

- Automated OCR fallback for unextractable PDF pages.
- Hybrid search (dense vector retrieval combined with exact term/acronym matching).
- Strict document-scoped vector isolation ('document_id' metadata filtering).
- Explicit citation tagging ('[C1]', '[C2]') tied to

Table -2.4.1: Existing Gap V/s Motivation

Existing Gap	Motivation/Proposed Solution
Traditional document search relies on keyword matching and cannot understand the context of user queries.	Develop a semantic search system that understands the meaning of queries and retrieves contextually relevant information.
Users spend significant time manually searching through lengthy documents to find specific information.	Build an AI-powered assistant that provides instant answers through natural language interaction, reducing search time and effort.
Existing AI document assistants often have limited support for secure processing of custom uploaded documents.	Design a secure platform that allows users to upload and query their own documents while ensuring privacy and efficient processing.
Most existing solutions do not integrate document parsing, semantic retrieval, and LLM-based answer generation into a single workflow.	Develop a unified system combining NLP, RAG, ChromaDB, and Google Gemini API to deliver accurate, context-aware responses.

3. METHODOLOGY

The system architecture of DocAssist AI follows a decoupled multi-layer client-server paradigm comprising a ReactJS Frontend, a Flask REST API Backend, a ChromaDB Vector Store, a SQLAlchemy Database, and the Google Gemini API.

The is as follows:

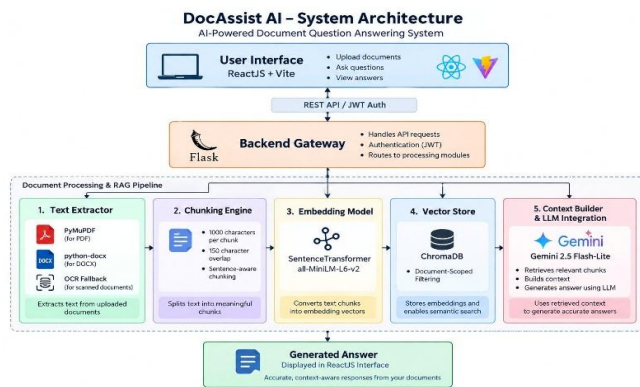


Fig - 3.1: Overall workflow of the proposed system

3.1 Document Ingestion & Text Parsing

When a user uploads a document, the backend validates the file type and passes it to the extraction pipeline:

- **PDF Processing**: Handled via PyMuPDF (`'fitz'`). If page text is empty (scanned or image-based PDF), the system triggers an OCR Fallback using `'pytesseract'` and `'PIL'` to extract text from rendered page bitmaps.
- **DOCX Processing**: Parsed via `'python-docx'`, extracting paragraph text and tabular data cells.
- **TXT & Web URLs Ingestion**: Cleaned plain text parsing; public web pages are scraped and sanitized using `'BeautifulSoup'` with SSRF security checks.

3.2 Paragraph & Sentence-Aware Text Chunking

To prevent slicing through words or breaking semantic coherent units, the extracted text is processed using a boundary-aware chunking algorithm:

Chunk Size = 1000 characters
Overlap = 150 characters

Paragraphs are split at double-newlines (`'\n\n'`), followed by sentence segmentation (`'[.!?]'`). Chunks are constructed incrementally until the character threshold is reached, retaining 150 characters from preceding boundaries to maintain contextual continuity.

3.3 Dense Vector Embedding & Vector Database Indexing

Each chunk c_i is mapped to a 384-dimensional vector embedding using `SentenceTransformer("all-MiniLM-L6-v2")`:

$$v_i = \text{Embed}(c_i) \in \mathbb{R}^{384}$$

The chunks, embeddings, and associated metadata (document_id, user_id, chunk_index) are stored in a persistent ChromaDB collection (docassist_docs).

3.4 Hybrid Semantic & Keyword Retrieval

Given a user prompt q , the query vector $v_q = \text{Embed}(q)$ is generated. DocAssist AI queries ChromaDB using document-scoped filtering:

$$\text{Top-k Chunks} = \arg \max_{c_i \in D_{\text{doc_id}_j}} \cos(v_q, v_i)$$

To prevent retrieval failure on technical acronyms (e.g., RAG, NLP, SRS, API), the retrieval pipeline includes a Hybrid Keyword Fallback. It extracts capital acronyms and non-stopwords from the prompt, scans document chunks for exact matches, and prepends them to the retrieved vector context window.

3.5 Context-Restricted Answer Generation & Citation Markup

For **Microsoft Word**, copy and paste this version directly: The top $k = 7$ retrieved chunks are concatenated into a structured context prompt:

Context:

[C1] Chunk 1 text...

[C2] Chunk 2 text...

Question: {user_question}

Instruction: Answer using **ONLY** the context provided above. If the information is missing, state that clearly. Cite sources using [C1], [C2], etc.

This payload is transmitted to the **Google Gemini API** (gemini-2.5-flash-lite / gemma-3-12b-it). The LLM returns a synthesized answer with embedded citation tags.

4. RESULTS

The performance and reliability of DocAssist AI were evaluated using functional testing, execution latency analysis, and answer accuracy checks across various document types.

Input:

Uploaded file – "AI Research Overview.pdf"

Query – "What are the key advantages of using Retrieval-Augmented Generation?"

System Process:

File parsed → **Embeddings generated** → **Query processed** → **Relevant chunks retrieved**

Output:

"Retrieval-Augmented Generation (RAG) improves the factual accuracy and context awareness of AI models by combining real-time document retrieval with large language model reasoning. It ensures that generated answers are

grounded in the provided document context, making responses more reliable and domain-specific.”

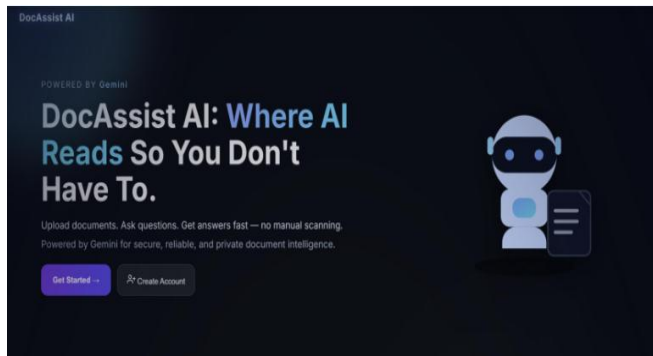


Fig - 4.1: Home Page

4.1 System Performance & Latency Metrics

Tests were conducted on representative documents (average 15 pages per PDF, ~6,000 words):

- **Average Text Extraction Time:** 0.42 s (Native PDF), 2.85 s (Scanned Page OCR).
- **Embedding & Indexing Time:** 1.15 s for ~40 chunks.
- **Vector Retrieval Latency:** 0.08 s (ChromaDB top-7 cosine search).
- **LLM Generation Response Time:** 1.40 s – 2.10 s (Gemini API).
- **Total End-to-End Query Latency:** $\approx$ 1.5 s – 2.3 s.

4.2 Functional Verification Results

1. **PDF Extraction:** Standard text PDFs ranging from 10 to 50 pages were tested. The system successfully extracted the text cleanly using PyMuPDF. The test was marked as Pass.
2. **Scanned PDF Handling:** Image-only PDF pages were tested to evaluate scanned document processing. The system automatically triggered OCR using Tesseract and successfully extracted the required text. The test was marked as Pass.
3. **DOCX / TXT Extraction:** Multi-table DOCX and TXT files were used as input to test document extraction. The system successfully parsed the text and table content into structured text. The test was marked as Pass.
4. **Chunking Engine:** A document containing approximately 100,000 characters was used to evaluate the chunking process. The system generated approximately 110 boundary-aware chunks while maintaining contextual information. The test was marked as Pass.
5. **Vector Storage:** ChromaDB insert operations were tested to verify vector storage and indexing. The generated embeddings were successfully indexed under the docassist_docs collection. The test was

marked as Pass.

6. **Document Isolation:** Document isolation was tested by querying Document A while Document B was active. The system prevented cross-document data leakage by enforcing the document_id filter. The test was marked as Pass.
7. **Question Answering:** The system was tested using both conceptual and factual questions. It generated precise answers based on the retrieved document context and provided citations using [C1] references. The test was marked as Pass.
8. **Authentication and Security:** Invalid JWT tokens and unauthorized document access requests were tested. The system correctly denied unauthorized access and returned appropriate 401 Unauthorized or 403 Forbidden responses. The test was marked as Pass.

4.3 Document Upload, Extraction & Paragraph-Aware Chunking Pipeline

DocAssist AI processes multi-format files (PDF, DOCX, TXT) and web contents. Rather than using crude fixed-character slicing, the system employs paragraph- and sentence-aware chunking with sliding context overlap.

Implementation & Results:

- **Format Validation:** Restricts input to supported extensions (PDF, DOCX, TXT) with max size verification (25 MB).
- **Extraction Efficiency:** Standardized text extraction preserves paragraph boundaries and formatting semantics.
- **Chunking Configuration:** Target chunk size of 500 characters with a 50-character sliding overlap, balancing contextual density and retrieval specificity.

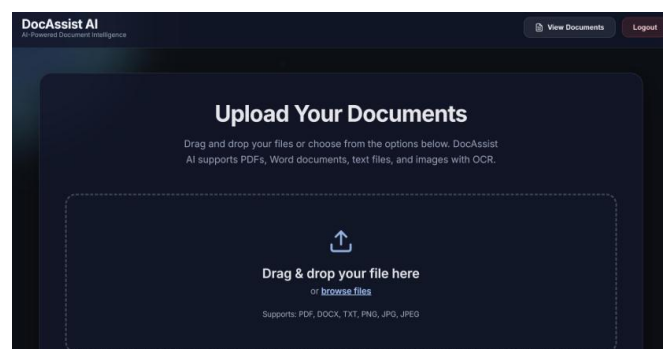


Fig - 4.3.1: Document Uploading Page

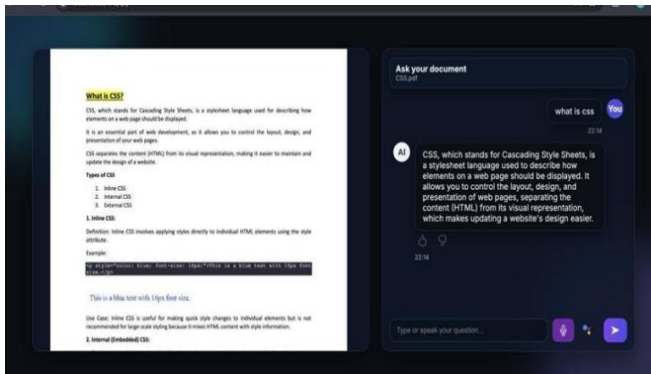


Fig - 4.3.2: Example with a pdf uploaded Page

4.4 User Authentication & Multi-Tenant Workspace Dashboard

The platform enforces secure multi-tenancy and workspace isolation through JSON Web Token (JWT) authentication. Upon successful login, users are granted access to a personalized workspace dashboard displaying their uploaded document repository, indexing statuses, and quick action controls.

Implementation & Results:

- **Authentication:** JWT tokens signed with SHA-256 HMAC, verifying user ownership on every API request.
- **Workspace Isolation:** Documents and vectors are strictly scoped by user_id metadata to prevent cross-tenant data leakage.
- **Dashboard Interface:** Provides real-time metrics including total documents indexed, vector count, and per-document processing status

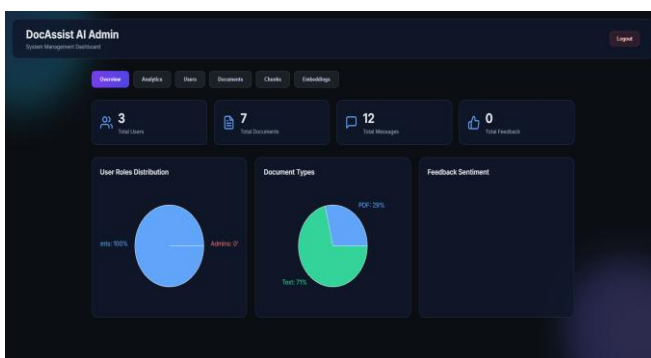


Fig - 4.3.2: Example with a pdf uploaded Page

5. DISCUSSION

The empirical implementation of DocAssist AI illustrates the practical advantages of Retrieval-Augmented Generation (RAG) in document assistance applications.

Elimination of Hallucinations: Standard baseline LLMs may generate inaccurate statements when asked about domain-specific documents. By restricting Gemini's prompt scope exclusively to the relevant context chunks retrieved from ChromaDB, DocAssist AI improves factual fidelity and ensures that generated responses are grounded in the available document content.

Value of Hybrid Retrieval: Dense vector embeddings may occasionally assign low similarity scores to unique technical acronyms or specific serial codes. Integrating exact-keyword matching alongside dense vector search helps resolve such query failures and improves retrieval for domain-specific terminology.

Importance of OCR Fallback: Standard PDF parsers may return empty text for image-based or scanned PDF documents. DocAssist AI addresses this limitation by automatically detecting pages with no extractable text and using pytesseract as an OCR fallback, thereby improving the system's robustness across documents with different formats and qualities.

Transparency Through Citations: The inclusion of source tags such as [C1] and [C2] allows users to verify generated answers against the corresponding passages from the original document. This improves transparency and helps establish greater trust in the system for academic and professional document-analysis workflows.

6. CONCLUSION

DocAssist AI successfully demonstrates a robust, end-to-end RAG architecture for document question answering. By integrating ReactJS, Flask, PyMuPDF, Tesseract OCR, ChromaDB, SentenceTransformers, and Google Gemini, the system provides a seamless platform for uploading documents and receiving instant, citation-backed answers to natural language questions. The implementation enforces document isolation, boundary-aware text chunking, hybrid retrieval, and secure user access management, presenting a viable solution for overcoming manual document search bottlenecks.

6.1 Future Scope

- **Multi-Modal Parsing:** DocAssist AI can be extended to support multi-modal document parsing by expanding vector index coverage to embedded chart figures, tables, and architectural diagrams using vision-language models.
- **Interactive Citation Highlighting:** The system can be enhanced to provide interactive citation highlighting, where users can click on a citation such as [C1] in the frontend and directly view the corresponding page or paragraph from the original PDF.

- Multi-Document Synthesis: Future development can enable cross-document querying, allowing users to perform comparative analysis across multiple research papers or documents simultaneously.
- Advanced Re-ranking: Advanced retrieval techniques can be incorporated by implementing cross-encoder re-rankers, such as BGE-Reranker, after the initial vector retrieval stage. This can further improve the precision and relevance of the context provided to the language model.

7. REFERENCES

- 1) P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," Advances in Neural Information Processing Systems (NeurIPS), vol. 33, pp. 9459–9474, 2020.
- 2) A. Vaswani et al., "Attention Is All You Need," Advances in Neural Information Processing Systems (NeurIPS), vol. 30, pp. 5998–6008, 2017.
- 3) J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proc. NAACL-HLT, pp. 4171–4186, 2019.
- 4) N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in Proc. EMNLP-IJCNLP, pp. 3982–3992, 2019.
- 5) Google DeepMind, "Gemini: A Family of Highly Capable Multimodal Models," arXiv preprint arXiv:2312.11805, 2024.
- 6) J. Johnson, M. Douze, and H. Jégou, "Billion-scale Similarity Search with GPUs," IEEE Transactions on Big Data, vol. 7, no. 3, pp. 535–547, 2021.
- 7) PyMuPDF Documentation, "fitz: Python binding for MuPDF library," 2024. [Online]. Available: <https://pymupdf.readthedocs.io/>
- 8) ChromaDB Team, "Chroma: The AI-native open-source embedding database," 2024. [Online]. Available: <https://docs.trychroma.com/>