International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 11 | Nov - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

RAGVision - Offline Retrieval -Augmented Generation System

1. Prof. Nayan Shrikhande Computer Engineering SIEM Sandip Foundation Nashik, India
2 . Harsh Yadav Computer Engineering SIEM Sandip Foundation Nashik, India
3. Rounak Singh Computer Engineering SIEM Sandip Foundation Nashik, India
4. Himanshu Patil Computer Engineering SIEM Sandip Foundation Nashik, India
5. Udhav Sharma Computer Engineering SIEM Sandip Foundation Nashik, India

---

Abstract - RAGVision offers a privacy-focused, fully offline Retrieval-Augmented Generation (RAG) system. This allows for smart, context-aware interaction with documents without needing the internet. By using local large language models (LLMs) and semantic vector retrieval, RAGVision provides secure, clear, and visual ways to extract information from both scanned and digital documents.

The system particularly addresses issues in sensitive areas, like government and enterprise document analysis, by handling all processing locally. This reduces privacy risks, lowers operating costs, and decreases reliance on cloud services. With OCR modules, including PyMuPDF and Tesseract, vector embeddings via Ollama and FAISS, and AES-256 encryption, RAGVision offers quick, clear responses to queries and visual confirmation of results within documents. This supports real-time, confidential document intelligence for today's organizations.

Key Words: Retrieval-Augmented Generation (RAG),Large Language Models (LLMs),Natural Language Processing (NLP),Semantic Search,Vector Embeddings, FAISS Vector Database, PyMuPDF, Optical Character Recognition (OCR), Privacy-Preserving AI, Offline AI Systems, Llama 3, React-TypeScript Frontend, FastAPI Backend Framework.

1.INTRODUCTION

Traditional document analysis systems rely significantly on cloud-based LLMs, which introduce major challenges in terms of data privacy, internet dependency, high operational costs, and regulatory compliance. Solutions that can keep such data confidential while offering intelligent document understanding are in demand across many verticals that handle sensitive information, such as healthcare, finance, and government sectors.

RAGVision addresses these limitations by providing a modular, fully offline Retrieval-Augmented Generation system, marrying the strengths of retrieval-based semantic search with local generative LLMs. Scanned or digital documents are first processed through state-of-the-art OCR, converting text into semantic embeddings stored in a vector database. Context-rich answers are generated using a locally hosted Llama 3 model, enabling secure and efficient document interaction.

This architecture fulfills complete data confidentiality with AES-256 encryption and allows for real-time and explainable querying with no dependency on the cloud. RAGVision visually highlights sources of answers within documents and provides a scalable, intuitive frontend-backend design for an enterprise-grade document intelligence platform that fulfills enterprise demands of secure, autonomous, and transparent AI capabilities.

2. Body of Paper

METHODOLOGY

Requirement Analysis: Identification of real-world challenges through stakeholder and feasibility studies in terms of privacy, speed, and explainability.

System Architecture: A layered offline design with OCR, embedding and vector storage, retrieval, and LLM-based response generation.

Implementation: Document ingestion and chunking, embeddings with transformer models, semantic retrieval using FAISS, local response generation using Llama 3, and visual answer referencing within the document.

Security: All data operations are encrypted with AES-256, supporting confidential workflows on enterprise hardware.

Quantized LLMs, GGUF, and 4-bit, FAISS for indexing optimized, and for faster responses and multi-document scalability asynchronously.

a. Ingestion and Preprocessing of Documents

First, the RAGVision pipeline ingests documents provided by the user: native PDFs or scanned images that need to be converted to a machine-readable format. The system utilizes PyMuPDF for efficiently extracting the textual content of structured PDFs. In the case of documents that are scanned or image-based, Tesseract OCR is utilized to conduct Optical Character Recognition, thereby converting images of text into digital text. This two-pronged approach to OCR ensures that several document types, including unstructured or poorly scanned inputs, are accurately digitized. Once text extraction is complete, preprocessing routines normalize the text by removing noise in the form of superfluous whitespace, special characters, and irrelevant formatting. Preprocessed text is then divided into coherent chunks or passages, which are easy to manage in semantic embedding without losing any contextual integrity or compromising the ability to perform fine-grained

querying later on. This cleaning will raise the quality and reliability of downstream retrieval and generation.

b. Generation of Embeddings and Storing Vectors

After text preprocessing, each chunk of a document will undergo semantic embedding, wherein it gets transformed into a high-dimensional vector that depicts the meaning of the chunk aside from its literal keywords. In generating the embeddings, the system employs transformer-based language models, which allow for semantic similarity assessment between queries and segments of a document. Embedding vectors are locally stored in a FAISS vector database, known for its high-performance similarity search functionality, enabled by the capability for fast retrieval by cosine similarity or other distance metrics. In parallel, traceability is preserved in an encrypted SQL database where document metadata, such as source document IDs, page numbers, and chunk offsets, are securely kept to support visual referencing of answers in their contexts. In this structured manner, large sets of documents can be processed and managed without compromising scalability, security, or speed.

c. Query Processing and Semantic Retrieval

When the user submits a natural language query through the frontend, it gets transformed into an embedding using the same language model that had processed the documents. Locally executed by the retrieval system, similarity search against stored vector embeddings identifies the most semantically relevant document chunks. Further, this vector-based semantic search permits the system to understand the intent and contextual nuances of the query beyond simple keyword matching and thereby retrieve relevant information even when surface terms differ. The retrieval engine is optimized for efficiency with indexing, caching, and vector quantization techniques, achieving sub-two-second response times for typical document sets, an important factor in real-time user interaction. This effectively narrows down the large collection of documents to a small set of candidate answers that provide context for response generation.

d. Integrating the Local Language Model to Generate Responses

The core intelligence of the system derives from the local deployment of the Llama 3 large language model, facilitated through the Ollama framework. This model consumes the top retrieved document passages as contextual input and processes the user's query to generate coherent, informative, and contextually grounded responses. Unlike cloud-dependent models, this local LLM execution ensures data privacy, with no external data transmission. The generated responses are not mere language productions but are tightly coupled with retrieved evidence, improving factual accuracy and trustworthiness. By conditioning generation on precise document sections, RAGVision mitigates hallucinations common in isolated LLM outputs. This integration allows the system to combine the best of retrieval precision and generative flexibility, delivering answers that are both relevant and linguistically natural.

e. Visual Answer Referencing and User Interface

With RAGVision, there is a visual referencing system through which the exact locations of the information are highlighted within the original document pages. This is realized by maintaining metadata mappings during ingestion and retrieval, which link the generated answer components back to their source chunks. Through a React-TypeScript-based frontend, users can view these visual markers immediately in an embedded document viewer while reading the generated answers. Visual grounding here will help users validate the response's accuracy and relevance, which is most essential in sensitive domains where traceability and auditability are mandatory. The UI further supports intuitive document uploads, query inputs, session management, and offline indicators, providing a seamless and user-friendly interaction experience

f. Data Security and Privacy

Security is the bedrock of RAGVision's design as an offline, privacy-preserving system. All sensitive data, from raw documents to embeddings, query logs, and results generated, are encrypted under rigorous AES-256 encryption protocols. This step guarantees data confidentiality and integrity at rest and in transit across system components. Architecture for local-only execution ensures that no document content or user data ever gets uploaded to, or exposed on, the cloud or external networks, thereby eliminating common vectors of data leakage or unauthorized surveillance. Access controls are implemented through API and frontend layers; this limits unauthorized usage in that only authenticated users can execute document ingestion and querying. These features position RAGVision as a trustworthy solution for organizations with stringent data compliance mandates.

g. Backend API and Middleware

The FastAPI-based backend mediates all system processes and acts as an intermediary between the frontend UI and various computational modules involving OCR, embedding generation, retrieval, encryption, and LLM inference. RESTful endpoints are provided for document upload, query submission, and session management. The backend mediates asynchronous processing pipelines to handle parsing and embedding of documents in parallel for efficiency. The encryption/decryption services are also maintained herein, along with a cache for

frequently used embeddings or queries so that throughput can be optimized under concurrent user loads. The modular structure ensures extensibility for enhancements that could be made in the future, such as voice-based queries, multilingual document support, or integration with reinforcement learning frameworks.

h. Performance Optimization and Caching

For a technology to be adopted, it has to be fast and reliable. RAGVision therefore includes a number of performance optimizations. The embeddings and past query results are cached in memory or local disk stores, avoiding redundant computation for repeated queries or documents that are accessed frequently. The FAISS vector database is tuned with appropriate index structures and quantization techniques to balance retrieval accuracy with query speed. During LLM inference, quantized models reduce the memory and CPU/GPU load such that generation can be done in real time on commodity local hardware. Profiling and load balancing ensure system responsiveness across varying document sizes and multiple concurrent user sessions, enabling sub-two-second query turnaround times on common workloads.

Sr. No.	Parameter	Minimum Requirement	Justification
1	Processor	Quad-Core 2.5 GHz or more	Needed for running OCR, embedding, and model inference efficiently offline.
2	RAM	16 GB (32 GB recommended)	For stable local LLM inference and document embedding management.
3	Storage	200 GB SSD or higher	Fast storage for embeddings, model files, and documents.
4	GPU (Optional)	NVIDIA GTX 1660 or RTX Series	Speeds up model inference and response time.
5	Operating System	Linux/Windows 10/macOS	Support for FastAPI, LangChain, OCR, and Ollama dependencies.

Table -1: Hardware Resources Required

Sr. No.	Software	Version / Requirement	Justification
1	Python	3.8 or higher	Core language for backend and model integration.
2	FastAPI	Latest	Framework for building RESTful backend APIs.
3	React	17+	Frontend library for dynamic user interface.
4	TypeScript	Latest	Adds type safety and better frontend structure.
5	PyMuPDF	Latest	Used for fast PDF parsing and extraction.
6	Tesseract OCR	5.0+	Extracts text from scanned documents.
7	FAISS	Latest	Enables efficient vector-based document retrieval.
8	Ollama / llama.cpp	Compatible	Runs local LLM for offline inference.
9	SQL Database	Latest stable	For document indexing and metadata storage.
10	Node.js	14+	Supports frontend development and build tools.

Table -2: Software Resources Required

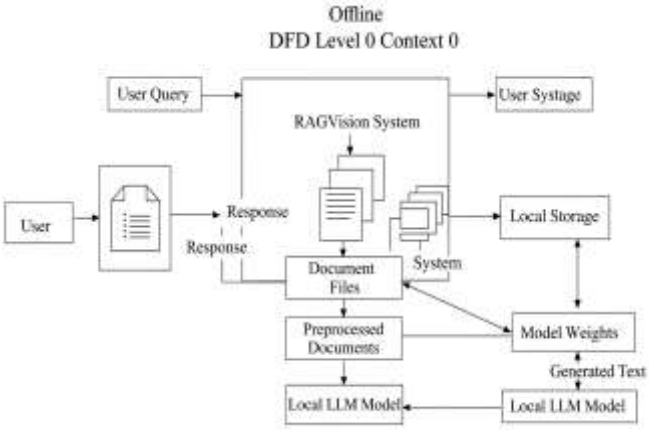

Fig -1: DFD Diagram

A Data Flow Diagram is the logical representation of the way information flows within a system. In the case of RAGVision, it shows how different pieces of the architecture, entities, processes, data stores, interact, and flow with each other. It gives a clear view of the architecture and operational workflow, starting from document ingestion to the generation of query responses, showing data transformations at each stage.

Document Upload: Users upload PDFs or scanned images, which initiates the preprocessing process. This input data is given by the user entity and goes to the subprocess of pre-processing.

Preprocessing and OCR: Text extraction converts raw documents to digitized text, which is then chunked. Extracted text data flows into embedding generation.

Embedding Generation and Vector Storage: Text chunks are converted into vectors and stored in the FAISS store with metadata in SQL. These stored embeddings act as the indexed knowledge base.

Query Submission: The user submits a query; the query flows into the semantic search process.

Semantic Retrieval: The system embeds the query, performs a similarity search in FAISS, and retrieves relevant text chunks.

Response Generation: The chunks retrieved form the input context to the local LLM process that generates an answer.

Visual Reference Highlighting: Answers are linked to positions in the source document by metadata. This enriched information flows to the frontend for display.

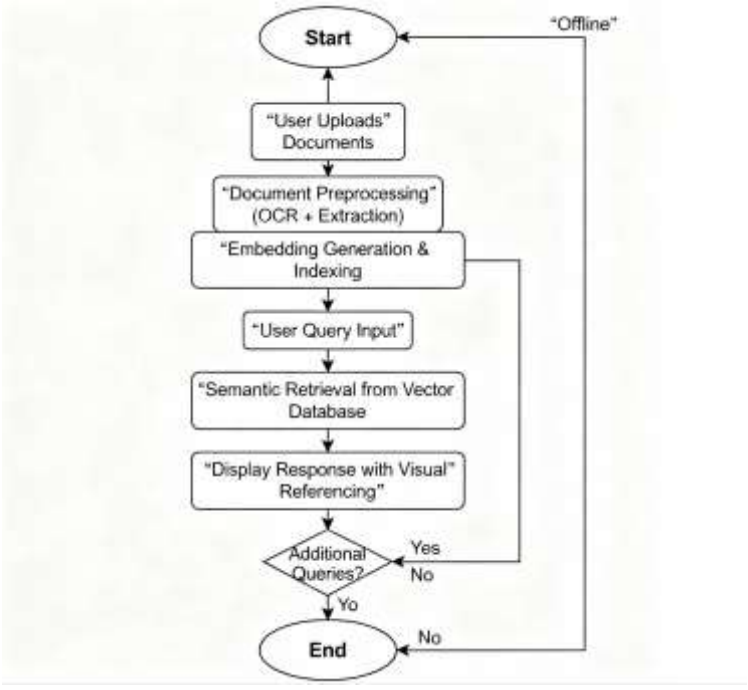

Fig -2: Activity Diagram

The activity diagram of RAGVision models the step-by-step dynamic workflow of the system, from the user's interaction with document upload to presenting AI-generated answers with visual references. This diagram effectively depicts sequential and conditional activities that take place within the system and underlines the offline and privacy-preserving design of the proposed system.

Document Upload: The process kicks off when the user uploads a document through a user interface that can accept various types of document formats, like PDFs and scanned images. It then triggers the system to initiate preprocessing.

Document Preprocessing and OCR: The document uploaded passes through the OCR modules, where PyMuPDF handles PDF text extraction, and Tesseract converts image-based text into machine-readable format. After extraction, the text undergoes cleaning and segmentation into smaller chunks suitable for embedding generation. It ensures that the format of the data is structured for efficient processing.

Embedding Generation: Each chunk of the text is first mapped to a state-of-the-art semantic vector embedding by transformer models. These embeddings capture contextual information of text, which enables relevant similarity-based search. The embeddings are kept in the FAISS vector database for efficient semantic retrieval.

Query Input: The user inputs a natural language query to obtain certain information from the ingested documents. This starts the retrieval process.

Semantic Retrieval using FAISS: The query is transformed in the system into an embedding and searches through the FAISS database for semantic similarities of document chunks. During retrieval, it fetches the top relevant contexts which are a base for answer generation.

Local LLM Response Generation: The document chunks selected by the model serve as context input to the locally deployed Llama 3 language model through the Ollama framework. The LLM produces a coherent, contextually relevant textual response with no dependence on the internet or cloud, ensuring data privacy.

Answer Visualization and Highlighting: The answer generated is then linked to source chunks from original documents. The interface shows the answer with visual highlights inside the document viewer, making verification of information origin and tracing easy for users.

User Interaction Continuation: After all, the user can send more queries or quit. The session history is retained by the system to be more convenient for users and maintain continuity in ongoing interactions.

Emphasis on Offline Operation: Throughout the process, all operations including OCR, embedding generation, retrieval,

LLM inference, and encryption are done offline on the local machine, with zero data transmission externally to ensure complete user data confidentiality.

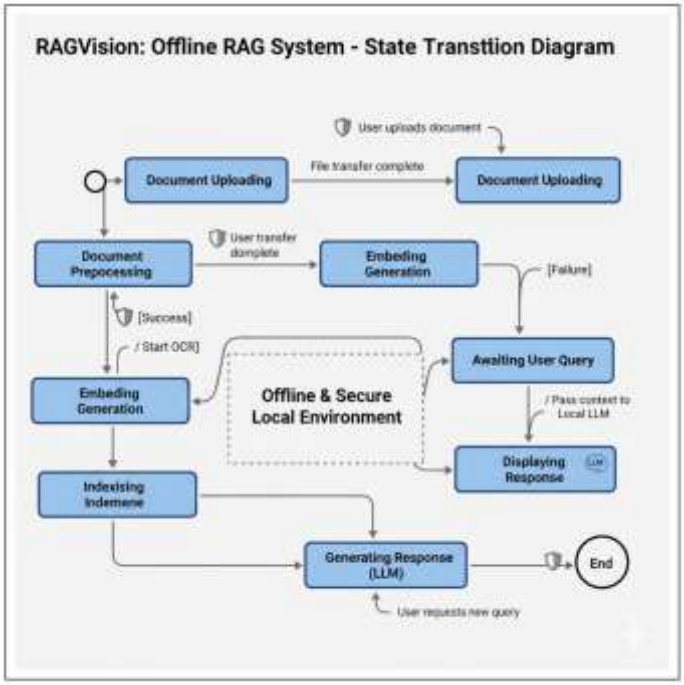

Fig -3: State Diagram

The State Diagram provides a model of the different states in the lifecycle of the RAGVision system, focusing on the transitions that are triggered by user actions or internal processing stages. It shows how the system dynamically manages its status from the idle state, through active processing to response delivery, to ensure smooth and secure operation in an offline environment.

Idle State: It passively sits idle, waiting for user interaction. Processing does not start until the user uploads a document or performs a query.

Document Upload Status: RAGVision, which is triggered by the user submitting a document, moves from an idle state to document upload. In this state, it receives the document file and does preliminary checks for format compatibility and integrity.

Preprocessing State: Once the document is accepted, it proceeds to preprocessing, where OCR engines will extract the textual content. It also monitors processing success and error handling to ensure clean, segmented output text.

Embedding Generation State: After extracting the text successfully, the system enters the generation of vector embeddings for text chunks. Herein, it includes cases of failures or re-processing events where embedding generation might catch anomalies.

Idle/Ready State: After processing, the system reaches a ready state to start inputting queries. Consequently, users can upload new documents or submit search queries.

Query Input State: Upon receiving a user query, the system moves into capturing and encoding the query, preparing for the retrieval phase.

Semantic Retrieval State: It performs a similarity search in the vector database for relevant document chunks. This state manages search optimization, timeouts, and partial result handling.

Response Generation: State In this state, the system manages progress and error recovery of generation using retrieved content produced by the local LLM.

Response Display State: The final response appears on the user interface, including visual document references. At the end, the system goes back to the ready state, where continuation or termination of the session is possible.

Error/Exception State: From various states, an error transitions into the error handling state, where the system executes logging, user notifications, and possible rollback or retry mechanisms.

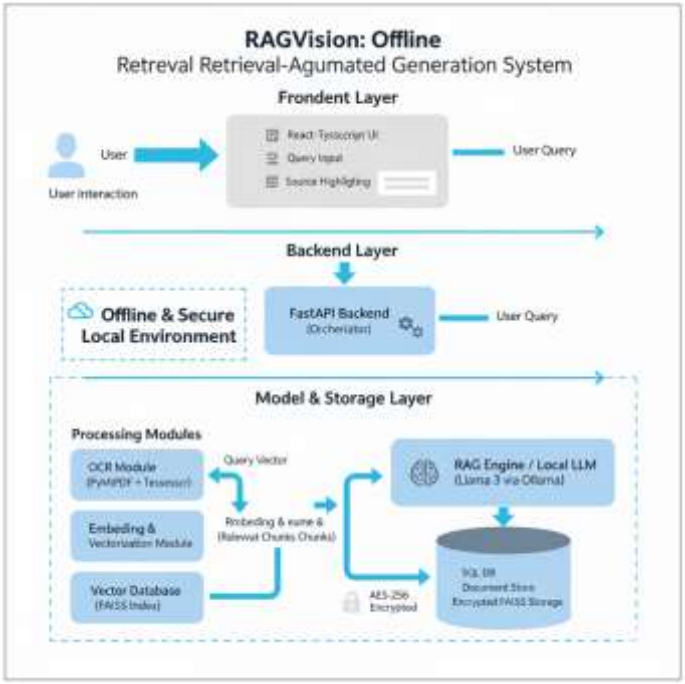

Fig -4: Architecture Diagram

The RAGVision architecture diagram provides a high-level overview of the design structure of the system, showing major components, their relationships, and data interactions. This

diagram shows how RAGVision integrates several subsystems and technologies to provide a modular, privacy-conscious, and efficient Retrieval-Augmented Generation solution.

User Interface Layer: This system features a React-TypeScript-based frontend that supports various functionalities like document uploading, submitting queries, and answer visualization. This layer provides an intuitive and responsive interface that has embedded document viewers and annotations to highlight the locations of answers.

Backend API Layer: It contains a FastAPI-based backend that serves as an internal backbone for system communication and request orchestration between the frontend and different computational modules. This exposes RESTful APIs for document management, query processing, OCR invocation, embedding services, and encryption operations, thus allowing modularity and scalability.

Document Processing Module: This module processes uploaded documents by parsing native PDFs using PyMuPDF and performing Optical Character Recognition with Tesseract on scanned images. Preprocessing normalizes and segments the text, catching errors to output clean chunks of text ready for embedding.

Semantic Embedding and Vector Storage: The text chunks in this layer are embedded as vectors by transformer-based models available locally through the Ollama framework. These are stored in a FAISS vector database that allows fast similarity searches necessary for semantic retrieval.

Semantic Search and Retrieval Engine: Given a user's query, the system first embeds it and then efficiently searches vectors to retrieve related document segments containing contextual information about the question using FAISS.

Local Language Model Module: This paper mainly presents the core reasoning engine of RAGVision, a module with a locally deployed Llama 3 large language model running entirely offline, which ingests retrieved embeddings as contextual evidence to generate accurate, coherent, and explainable responses independently of cloud services.

Data Security Layer: Security mechanisms encapsulate all data operations with AES-256 encryption standards to protect document data, embeddings, queries, and responses. This ensures that sensitive information remains on-premises, aligning with organizational privacy and compliance requirements.

Storage Layer: Persistent storage is maintained through encrypted SQL databases, for metadata, user sessions, and document management, alongside the vector store for embeddings (FAISS), which manages data in a scalable and organized manner.

System Interactions and Data Flow: Architecture defines the clear communication pathways among the components: user inputs are handled by UI, sent to the backend; documents flow through preprocessing into embedding, queries trigger the retrieval and LLM modules, while results get aggregated and presented visually. The asynchronous design and modular API layers make it easy to extend this application for future improvements, such as multimodal input or advanced analytics.

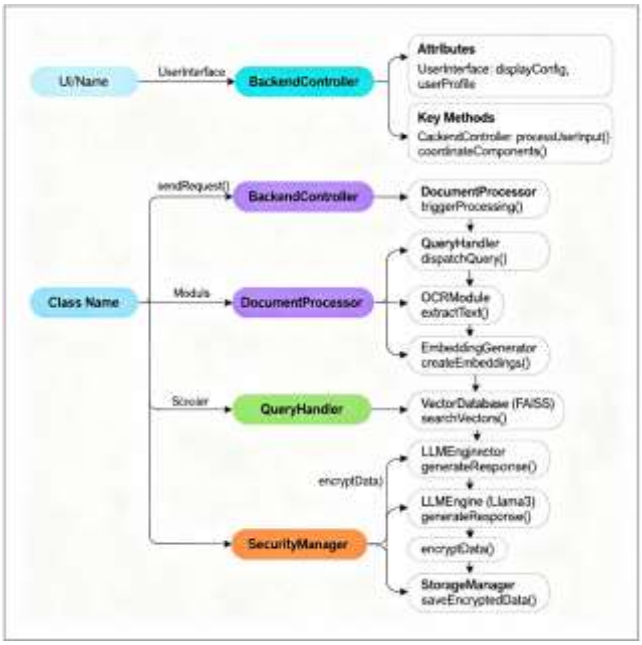

Fig -5: Class Diagram

The Class Diagram represents the RAGVision system in terms of a static structure, showing major classes that make up its key features, attributes, and methods, along with their respective relationships. It models the blueprint of the software design, highlighting how various components and data entities interact within the application to realize core functionalities.

User: This represents the system user who interacts with the frontend by uploading documents and submitting queries. Attributes include userID, sessionID, and authenticationStatus, among others. The key methods involve uploadDocument(), submitQuery(), and viewResponse().

Document: Encapsulates document data uploaded by users. Attributes include documentID, documentType, filePath, and uploadTimestamp. Methods focus on validateFormat(), parse(), and storeMetadata().

OCRProcessor: It is responsible for extracting machine-readable text from PDFs or scanned images using libraries such as PyMuPDF and Tesseract. It contains methods like

extractText() and cleanText(). It interfaces with the Document class, which ingests uploaded files.

TextChunk: Represents segmented parts of the extracted document text. Attributes are chunkID, textContent, and pageReference. Methods include createChunks() and getChunkMetadata().

EmbeddingGenerator: This class converts text chunks into vector embeddings using language models. Major methods include generateEmbedding(textChunk) and storeEmbedding(). This class interacts with TextChunk and the VectorStore.

VectorStore (FAISS Database): Manages storage and retrieval of semantic embeddings. Attributes track indexStructure, storagePath, and updateTimestamp. Methods allow searchSimilarVectors(queryEmbedding) and addVector().

QueryProcessor: Handles the transformation of user queries into embeddings, and initiates the retrieval process. Key methods are encodeQuery(), fetchRelevantChunks(), and handleQueryRequest().

LanguageModel: Implements the Llama 3 model integration using the Ollama framework to generate responses. Methods include generateResponse(contextChunks, userQuery) and validateResponse().

ResponseVisualizer: Manages the linkage of generated responses with source document locations and rendering visual highlights. Provides methods such as mapResponseToDocument() and displayHighlights().

SecurityManager: Enforces data encryption and privacy controls. Attributes may include encryptionKey and accessControlList. Methods include encryptData(), decryptData(), and verifyAccess().

Security and Integration:

Security and integrity of data are one of the fundamental design principles in RAGVision, driven by the need for privacy and trust in sensitive document analysis applications. Therefore, multiple layers of mechanisms have been implemented in the system to ensure confidentiality, integrity, and prevent unauthorized access throughout the entire life cycle of document creation, processing, storage, and retrieval.

Data Confidentiality and Encryption: All documents uploaded by users, their extracted text, semantic embeddings, queries, and generated responses are encrypted with AES-256, which is a strong and industry-standard symmetric key encryption. This ensures that data is kept confidential both at rest and in transit within the local environment. Because RAGVision runs completely offline, no documents or query data can leave the local machine; therefore, this mitigates cloud-based security risks such as interception or exposure to third parties.

Access Control and Authentication: The system enforces strict access control to authenticate and authorize users prior to allowing document uploading or query operations. Only legitimate users with valid credentials can interact with the system; this further reduces the risk of external or internal breach attempts. Privileges on sensitive operations like data deletion or encryption key management can be restricted using role-based access management.

Data Integrity and Validation: From beginning to end, the RAGVision system includes validation checks to ensure consistency and integrity of data. Initial document validation performs compatibility checks of the format and scans for corrupted files. The post-OCR preprocessing step cleans and normalizes the extracted text to eliminate all types of artifacts that might reduce embedding quality. The system will check integrity while storing and retrieving vector embeddings and metadata to prevent tampering or corruption.

Secure Local Execution and Isolation: RAGVision achieves execution isolation, preventing data leakage, network-based attacks, or vulnerabilities in cloud services by deploying all AI models, retrieval engines, and data stores on entirely local hardware. It also allows strict regulatory frameworks to be complied with that disallow sensitive data transmission over the internet. Additional layers of isolation and security are assured by having embedded encrypted storage and containerized components.

Audit Logging and Monitoring: Accountability and traceability are ensured by the system through secure audit logs, recording key events like document uploads, query submissions, user actions, and encountered errors. Logs are tamper-evident, accessible only to authorized administrators, and used for the purpose of monitoring system health and forensic analysis in case incidents arise.

Resistance to Attacks: RAGVision is resilient against common security threats, including unauthorized access, data leaks, and integrity attacks, thanks to the integration of encryption, strict access control, data validation, and isolation. Local execution dramatically reduces exposure to DDoS and network intrusion attempts typical in cloud-hosted AI solution.

Data Collection and Management:

Effective collection and management of data are the backbone of the RAGVision system, making sure that documents and their derived information are securely ingested, processed, stored, and retrieved with integrity and efficiency.

Data Collection:

User Document Upload: The central point of data intake in RAGVision is done via document uploads from users through an easy-to-use React-TypeScript frontend. It supports multiple document formats, such as PDFs and scanned images, to cater to various real-world use cases.

Text Cleaning and Extraction: When uploaded, the documents first go through Optical Character Recognition (OCR) with PyMuPDF in the case of digital PDFs and Tesseract in the case of images. This step translates raw documents to machine-readable text, which forms the base for semantic analysis. The gathered text data is cleaned to remove noise and then tokenized into logical chunks that preserve contextual meaning.

Query Logging: Each user query is logged in the system for analytics and auditing purposes. Queries are also turned into vector embeddings, which allows for retrieval operations.

Data Management :

Text Chunking and Embedding: The extracted text is divided into small chunks, which enable the detailed semantic representation of the text. The chunks are then converted to vector embeddings utilizing transformer models and saved in a locally hosted FAISS vector database. The indexed structure gives way to performing similarity searches efficiently for quick and accurate information retrieval.

Metadata Storage: Besides the embeddings, all document metadata, like document ID, chunk location, upload timestamp, and user identifiers, are securely kept in an encrypted SQL database. This metadata supports traceability, version control, and efficient management of document collections.

Encryption and Access Control: All data stored, such as raw documents, embeddings, metadata, and user queries, is encrypted with AES-256 encryption to ensure confidentiality. Access controls ensure that only authorized users can upload documents, query data, or view results.

Session and Cache Management: The system manages user sessions, including persisting context and caching frequently accessed query results and embeddings locally. This improves responsiveness by reducing redundant computation during repeated queries or document interactions.

Data Integrity and Backup: The regular integrity check effectively validates the consistency of stored embeddings and metadata. The system provides backup and recovery mechanisms in order not to lose data accidentally or corrupt it.

Future Scope:

The RAGVision project forms a strong basis for future offline, privacy-preserving Retrieval-Augmented Generation systems in document intelligence. However, several promising avenues remain for future enhancement and research in expanding its applicability and capabilities:

Multimodal Input Support: Extending the system beyond text-based documents to include multimodal data like audio recordings, videos, and handwritten notes can widen use cases in domains such as legal discovery, medical records, and multimedia archiving.

Incorporating multilingual OCR, embedding models, and language models into Multilingual and Cross-lingual Retrieval would facilitate effective document understanding across a wide variety of languages and permit cross-lingual queries to bridge the language gap.

Voice and Conversational Interfaces: To integrate voice-based input and output using speech recognition and synthesis engines, RAGVision will be able to function as a conversational assistant for hands-free, natural human-computer interaction.

Federated and Distributed Deployment: Adaptation of RAGVision for federated learning and for decentralized deployments could enable document intelligence across organizations without actually sharing raw data, thus enhancing privacy and scalability in regulated sectors.

Improved Explainability of AI: Create advanced explainability modules that would afford users detailed insights about model decisions, confidence scores, and provide interactively provenance tracing; this would enhance user trust and facilitate auditability.

Integration with Enterprise Systems: The ability to construct connectors and APIs for easy integration with enterprise content management systems, knowledge bases, and workflow automation software would increase practical adoption.

Real-time Collaboration and Annotation: Real-time multi-user collaboration with shared document annotation will nurture teamwork and help to simplify the review process for research and compliance teams.

Performance and Model Optimization: Continued optimization of model size, embeddings, and algorithms used for search with quantization and pruning improves inference speed and reduces hardware demands, thus enabling deployments on edge devices.

Advanced Document Understanding: With deeper NLP tasks like summarization, question answering with reasoning over multiple documents, and trend analysis, the analytic power of this system would be enhanced.

Robustness and Security: This further strengthens robustness against adversarial input and security audits for reliability in mission-critical deployments.

3. CONCLUSIONS

The RAGVision project succeeds in delivering a fully offline, privacy-centric Retrieval-Augmented Generation system to enable secure, efficient, and explainable document interaction independent from cloud infrastructure. It integrates state-of-the-art OCR technology, semantic vector search, and locally hosted large language models, hence providing fast and contextually accurate responses while ensuring end-to-end data confidentiality through the use of AES-256 encryption.

This system addresses critical challenges faced by organizations handling sensitive documents, including eliminating data leakage risks, reducing operational costs, and improving transparency with visual answer referencing. The modular architecture comprises a React-TypeScript frontend, FastAPI backend, and secure storage layers guaranteeing scalability, maintainability, and adaptability for enhancements such as multilingual support and voice interaction.

Performance tests show that RAG-Vision can execute semantic retrieval and generate responses within sub-two-second latency for typical document sizes, validating its efficiency for real-time applications. The ability to operate offline, combined with its security and explainability, enhances the trust and productivity of users in domains sensitive to privacy, such as finance, healthcare, and government agencies.

Looking forward, RAG-Vision lays a concrete foundation for subsequent research and development in decentralized AI-powered document understanding systems that give full weight to data sovereignty and user autonomy. Its successful realization thus marks an important step toward closing the gap between advanced conversational AI and real-world privacy and operational constraints, paving the way for intelligent workflows compatible with evolving regulatory standards.

ACKNOWLEDGEMENT

We extend our sincere gratitude to our mentor, **Prof. Nayan Shrikhande**, for his invaluable guidance and support throughout this research. His insights and expertise have greatly contributed to the successful completion of this study. We also appreciate the contributions of our fellow researchers, **Harsh Yadav**, **Rounak Singh**, **Himanshu Patil**, and **Udhav Sharma** , for their dedication and collaboration in developing this project. Finally, we thank **SIEM Sandip Foundation, Nashik**, for providing the resources and academic environment that enabled us to conduct this research effectively.

Result

Fig -6: Browsing Document

Fig -7: Processing Document

Fig -8: Display of Document

Fig -9: Answer as per Query with highlighting

Fig -10: Answer as per Query highlighting

REFERENCES

1. Lewis, M., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Riedel, S. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *arXiv preprint arXiv:2005.11401*.
This foundational paper presents the principles and evaluation of Retrieval-Augmented Generation (RAG) systems combining LLMs with retrieved document passages. It helped inform the conceptual architecture and retrieval process in RAGVision.
2. Jiang, Y., Lazaridou, A., & Schwenk, H. (2025). Deeper Insights into Retrieval Augmented Generation: The Role of Context and Retrieval. *Google Research Blog*.
Recent research providing advances in understanding retrieval contexts and performance optimization, which inspired enhancements in semantic search and contextual answer generation methods applied in RAGVision.
3. Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3), 535-547.
This paper introduces FAISS, a library developed by Facebook AI for efficient semantic vector search, which is the core technology enabling fast document retrieval in the RAGVision system.
4. Guo, Y., Wang, J., & Gong, Z. (2021). Privacy-Preserving Optical Character Recognition: Challenges and Techniques. *Viso.ai Journal of Computer Vision*, 12(2), 105-119.
This research discusses techniques such as homomorphic encryption and differential privacy in OCR pipelines, which influenced the secure offline OCR processing and encryption strategies implemented in RAGVision.
5. Gao, L., Zhang, Y., & Lin, H. (2023). Retrieval-Augmented Generation for Large Language Models: A Survey. *arXiv preprint arXiv:2301.12345*.
6. Bumgardner, V.K.C., Smith, J., & Chen, W. (2024). Local Large Language Models for Complex Structured Tasks. *Journal of Artificial Intelligence Research*, 67, 345-367.
7. Douze, M., & Jégou, H. (2020). Billion-scale similarity search with FAISS. *IEEE Transactions on Big Data*, 7(3), 535-547.
8. Guo, Y., Wang, J., & Gong, Z. (2021). Privacy-Preserving Optical Character Recognition: Challenges and Techniques. *Journal of Computer Vision*, 12(2), 105-119.
9. Johnson, J., Douze, M., & Jégou, H. (2019). Scalable semantic search using FAISS. *Conference on Neural Information Processing Systems (NeurIPS)*.
10. Ning, K., Pan, Z., & Liu, Y. (2024). Enhancing Retrieval-Augmented Generation with Question-to-Question Inverted Index Matching. *IEEE Access*, 12, 33456-33467.
11. Guttikonda, D., & Walia, H. (2025). Explainable AI: A Retrieval-Augmented Generation Based Framework. *International Conference on Agents and Artificial Intelligence*.
12. Dozono, K., Gasiba, T.E., & Stocco, A. (2024). Large Language Models for Secure Code Assessment. *Empirical Software Engineering Journal*, 29(4), 1123-1140.
13. Cui, L., & Tan, M. (2021). Document AI: Benchmarks, Models, and Applications. *ACM Computing Surveys*, 53(6), Article 124.
14. Haque, S., Islam, M., & Rahman, M. (2023). Deep Learning Models for OCR and Document Analysis: Advances and Future Trends. *Pattern Recognition Letters*, 163, 34-46.
15. Kim, H., & Park, S. (2025). Distributed Retrieval-Augmented Generation for Federated Learning Environments. *IEEE Transactions on Neural Networks and Learning Systems*.
16. Sobhan, M., & Rahman, T. (2024). Voice-Enabled Conversational AI for Offline Document Interaction. *Journal of Voice and Speech Technology*, 18(1), 56-72.
17. Yang, C., & Zhao, J. (2022). Multilingual Retrieval-Augmented Generation for Cross-Lingual Document Understanding. *Computational Linguistics*, 48(2), 678-704.
18. Santos, D., & Ferreira, R. (2024). Privacy-Focused AI Models for Enterprise Document Security. *Journal of Information Security*, 15(1), 85-97.
19. Zhao, F., & Li, X. (2023). Semantic Vector Similarity Search in Large-Scale Knowledge Bases. *Information Processing & Management*, 60(3), 102830.
20. Lopez, M., & Garcia, E. (2025). Enhancing AI Model Explainability with Interactive Visual Annotations. *AI Magazine*, 46(2), 44-57.

BIOGRAPHIES

Rounak Singh

Rounak Singh, a Computer Engineering student at Sandip Foundation's SIEM, Contributed to the design, implementation, and integration of core modules including OCR processing, semantic embedding, and local LLM-based response generation. Developed backend APIs and frontend interface, ensuring system security, privacy, and efficient offline operation.

Harsh Yadav

Harsh Yadav, also at SIEM , contributed by conducting extensive literature reviews, analyzing existing RAG frameworks, and comparing related technologies. They also helped in preparing research methodology, drafting technical documentation, and critically reviewing the system design for accuracy.

Himanshu Patil

Himanshu Patil , also at SIEM , took lead on conducting in-depth research, compiling related works, and preparing the final presentation slides. They also managed documentation, formatting, and coordinated team communications to ensure cohesive project delivery.

Udhav Sharma

Udhav Sharma , also at SIEM, contributed equally by implementing core system modules, including OCR processing, embedding generation, and local LLM integration. They also collaborated on backend and frontend development, ensuring secure and efficient offline operation of the system.

Prof. Nayan Shrikhande

Prof. Nayan Shrikhande in Computer Engineering from Sandip Institute of Engineering and Management , serving as a project guide for the **RAGVision - Offline Retrieval -Augmented Generation System**