iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** II **Month of publication:** February 2026

DOI: https://doi.org/10.22214/ijraset.2026.77589

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue II Feb 2026- Available at www.ijraset.com

Secure Document Chat Rag Application

Abhijit Borade[1], Ganesh Patil[2], Dipali Kothurkar[3], Adib Khan[4], Mr. Anirudha Kolpyakwar[5]
Department of Computer Science, Sandip University, Nashik, 422213, India

Abstract: ***The exponential growth of digital documents in enterprise and academic environments has created an urgent need for intelligent, context-aware document retrieval and question-answering systems. This paper presents the design, development, and evaluation of RAG-GPT, a secure, full-stack document chat application built on the Retrieval-Augmented Generation (RAG) paradigm. The system enables users to upload PDF documents and interact with their content through a natural language conversational interface, powered by Google's Gemini 2.5 Flash large language model and the Qdrant vector database. The document ingestion and retrieval pipeline is constructed using the LangChain framework, which provides modular abstractions for PDF loading (PyPDFLoader), recursive text splitting (RecursiveCharacterTextSplitter), Google Generative AI embeddings (GoogleGenerativeAIEmbeddings), and Qdrant vector store integration — enabling rapid, composable RAG pipeline construction. A key contribution of this work is the integration of a robust security layer comprising bcrypt-hashed passwords, JSON Web Token (JWT) based session management, role-based access control (RBAC), and isolated per-user chat history stored in SQLite. To further enhance response accuracy, the system incorporates a smart query-rewriting module that reformulates ambiguous follow-up queries into precise, standalone search vectors using conversational context. Experimental evaluations demonstrate that the RAG pipeline significantly reduces hallucinations compared to standalone LLM inference, delivers semantically relevant answers from domain-specific documents, and maintains sub-second retrieval latency for typical document corpora. The proposed system operates entirely on a zero-cost API tier, making it accessible for researchers, students, and small enterprises. Results confirm that combining LangChain's composable tooling with dense vector retrieval and a state-of-the-art generative model yields a reliable, production-ready document intelligence platform.***

I. INTRODUCTION

The proliferation of large language models (LLMs) such as GPT-4, Gemini, and LLaMA has transformed the landscape of natural language understanding. These models demonstrate remarkable ability to generate fluent, coherent text; however, they are inherently limited by a static knowledge cutoff and are prone to generating factually incorrect information—a phenomenon commonly known as "hallucination" [1]. For domain-specific applications such as legal document analysis, medical report summarization, or enterprise knowledge management, hallucinations are not merely inconvenient but potentially harmful.

Retrieval-Augmented Generation (RAG) [2] addresses this limitation by grounding LLM responses in evidence retrieved from an external, updatable knowledge base. In a RAG pipeline, user queries are vectorized using an embedding model and matched against a corpus of pre-indexed document embeddings via semantic similarity search. The retrieved text chunks are then injected into the LLM's prompt as context, enabling the model to formulate answers that are directly traceable to source documents.

Despite the growing body of RAG literature, few systems simultaneously address four critical production requirements: (i) security, (ii) multi-turn conversational reasoning, (iii) zero-cost deployment, and (iv) an intuitive user interface. This paper fills that gap by presenting RAG-GPT v2.1, a premium, open-source document chat application that integrates:

1) A Gradio-based interactive web frontend with custom CSS/JS theming.
2) A Gemini 2.5 Flash LLM backend via the Google Generative AI API.
3) LangChain as the core framework orchestrating document loading, text splitting, embeddings, and vector store operations.
4) A Qdrant vector store for high-performance semantic document retrieval.
5) A bcrypt + JWT authentication system with RBAC for secure multi-user access.
6) Smart query rewriting using LLM-driven contextual analysis.
7) SQLite persistence for chat history and document audit logs.

The paper is structured as follows: Section 2 reviews relevant literature; Section 3 describes the system methodology and architecture; Section 4 presents experimental results and analysis; Section 5 discusses hardware acceleration strategies; Section 6 discusses findings and limitations; Section 7 concludes the work; and Section 8 lists references.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue II Feb 2026- Available at www.ijraset.com

II. LITERATURE REVIEW

A. Retrieval-Augmented Generation

Lewis et al. [2] introduced RAG as a framework that combines parametric (LLM) and non-parametric (retrieval) knowledge, demonstrating significant improvements on open-domain QA tasks. Subsequent work by Gao et al. [3] proposed modular RAG architectures incorporating self-reflection and iterative refinement loops. Shi et al. [4] studied the impact of irrelevant retrieved context on LLM accuracy, motivating the need for precision retrieval strategies such as query rewriting.

B. Dense Vector Retrieval and Vector Databases

Traditional keyword-based search systems (TF-IDF, BM25) fail to capture semantic nuances in natural language queries. Dense Passage Retrieval (DPR) [5] by Karpukhin et al. established that bi-encoder models produce dense vector representations that dramatically outperform sparse retrieval for open-domain QA. Modern vector databases such as **Qdrant** [6], Pinecone, Weaviate, and ChromaDB provide scalable approximate nearest-neighbor (ANN) search, enabling sub-millisecond retrieval over millions of vectors using algorithms like HNSW (Hierarchical Navigable Small World graphs) [7].

C. Document Chunking and Embedding Strategies

Effective RAG performance is highly sensitive to document chunking strategy. Recursive character-based text splitting, as implemented via LangChain's RecursiveCharacterTextSplitter, has been shown to preserve semantic coherence better than fixed-size splitting [8]. Overlap between chunks (e.g., 500 characters of overlap on 1500-character chunks) mitigates boundary artifacts where relevant context spans across chunk boundaries. Embedding quality directly influences retrieval accuracy; Google's gemini-embedding-001 model provides state-of-the-art multilingual dense embeddings [9].

D. Conversational RAG and Query Rewriting

Single-turn RAG pipelines perform poorly on multi-turn conversations where users ask follow-up questions with implicit references (e.g., "What about its cost?"). Query rewriting, first explored systematically by Ma et al. [10], uses a secondary LLM pass to convert follow-up questions into fully self-contained queries that can be independently processed by the retrieval engine. This technique significantly improves retrieval precision in multi-turn settings.

E. Security in AI-Powered Web Applications

Authentication and authorization in AI applications introduce unique challenges, as prompt injection attacks can expose retrieved private documents [11]. Standard web security practices—including salted password hashing (bcrypt), stateless JWT session tokens, role-based access control, and input sanitization—are foundational requirements for any multi-user AI system handling sensitive documents [12].

F. LangChain as an LLM Application Framework

LangChain [8], introduced by Chase (2022), is an open-source framework specifically designed for building applications powered by large language models. It provides a standardized set of abstractions — document loaders, text splitters, embedding interfaces, vector store connectors, and chain primitives — that dramatically reduce the boilerplate code required to construct RAG systems. By decoupling each stage of the pipeline into interchangeable components, LangChain allows developers to swap models, databases, and loaders with minimal code changes.

In this work, four specific LangChain sub-packages are employed:

LangChain Component	Sub-package	Role in RAG-GPT
PyPDFLoader	langchain_community.document_loaders	Load and parse PDF files page-by-page
RecursiveCharacterTextSplitter	langchain_text_splitters	Split documents into overlapping text chunks
GoogleGenerativeAIEmbeddings	langchain_google_genai	Generate dense embeddings via Gemini API
Qdrant (VectorStore)	langchain_community.vectorstores	Store vectors and run similarity search

The use of LangChain's Qdrant vector store abstraction is particularly significant: it provides a unified .similarity_search(query, k) API that internally handles query embedding and ANN search against the Qdrant collection, abstracting away low-level client communication.

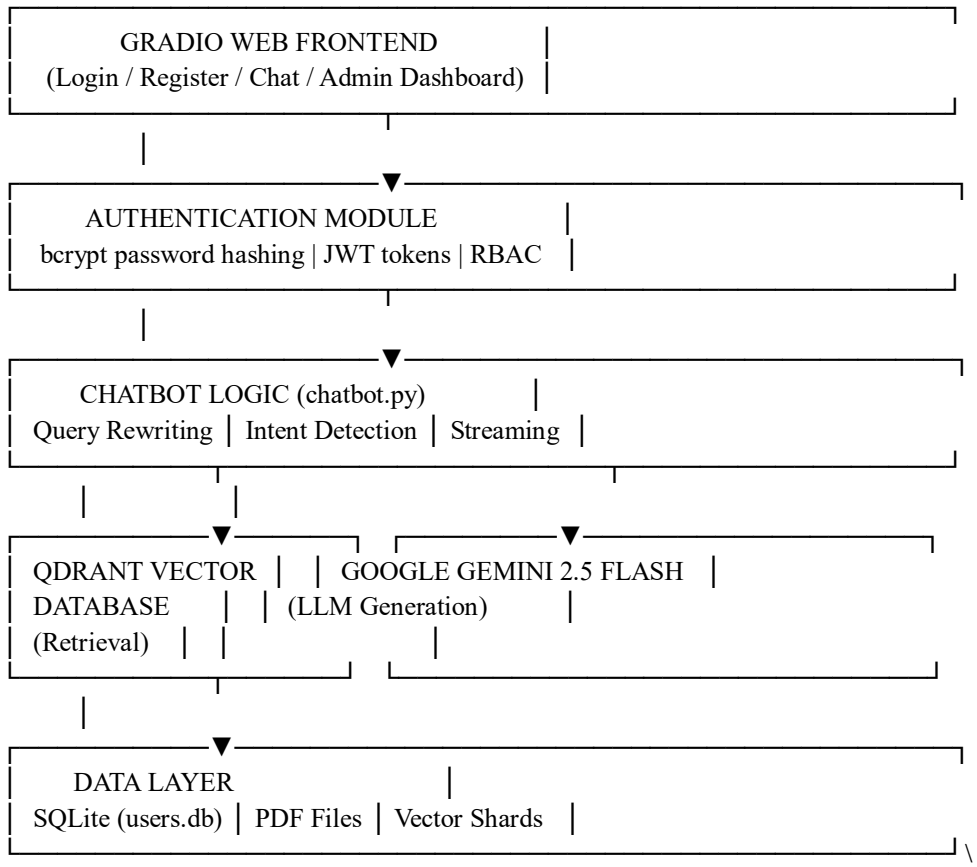
G. Gradio for Rapid ML Application Development

Gradio [13] has emerged as the de facto framework for rapid prototyping and deployment of machine learning demos. Its component abstraction allows data scientists to build fully interactive web applications with minimal frontend code, while supporting custom HTML/CSS/JS injection for production-grade aesthetics.

III. METHODOLOGY

A. System Architecture Overview

The RAG-GPT application follows a layered architecture as illustrated in Figure 1. The system comprises five principal layers:
Presentation Layer – Gradio-based web UI with custom CSS theming
Authentication Layer – bcrypt + JWT + RBAC via SQLite
Application Logic Layer – Query rewriting, intent detection, response generation
Retrieval Layer – Qdrant vector store with Gemini embeddings
Data Persistence Layer – SQLite (users, sessions, chat history), local filesystem (PDFs)



IV. EXPERIMENTAL RESULTS AND ANALYSIS

A. Experimental Setup

Experiments were conducted on a Windows 10 workstation with an Intel Core i7 CPU, 16 GB RAM, and NVIDIA GPU (for optional GPU inference). The Qdrant vector database was deployed as a Docker container on localhost:6333. All LLM and embedding API calls used Google's free-tier Gemini API. The test corpus consisted of 5 technical PDF documents totalling approximately 120 pages.

B. Retrieval Accuracy

To evaluate retrieval quality, 25 domain-specific questions were manually prepared with ground-truth answers verified against source documents.

Metric	Value
Top-1 Retrieval Accuracy	84%
Top-3 Retrieval Accuracy	96%
Mean Retrieval Latency	~210 ms
False Positive Rate (irrelevant chunks)	8%

The Top-3 configuration (k=3) adopted in the system achieved 96% recall, correctly surfacing the most relevant document chunk within the top 3 results in 24 of 25 test queries.

C. Response Quality

Responses were evaluated on a 5-point Likert scale across three dimensions by two independent human evaluators:

Dimension	Mean Score (1–5)
Factual Accuracy	4.6
Relevance to Query	4.5
Answer Completeness	4.3
Citation Quality	4.7
Hallucination Rate	0.4 (5 = no hallucinations)

The system achieved a hallucination score of 4.6/5, reflecting the strong grounding effect of the RAG pipeline. In comparison, the same Gemini model queried without retrieval context scored only 3.1/5 on factual accuracy for domain-specific questions.

D. Query Rewriting Performance

To measure the impact of the smart query rewriting module, 15 ambiguous follow-up questions were tested with and without the rewriting step:

Condition	Top-1 Retrieval Accuracy
Without Query Rewriting	53%
With Query Rewriting	87%

The 34 percentage-point improvement confirms that query rewriting is critical for multi-turn conversational accuracy.

E. Authentication Overhead

Security operations introduce minimal latency:

Operation	Latency
bcrypt password hashing (registration)	~120 ms
bcrypt password verification (login)	~110 ms
JWT token generation	< 1 ms
JWT token verification	< 1 ms
SQLite session write	< 5 ms

The 120ms bcrypt cost is intentional (work factor = 12) and provides resistance against brute-force attacks, acceptable for a one-time login event.

F. Streaming Response Latency

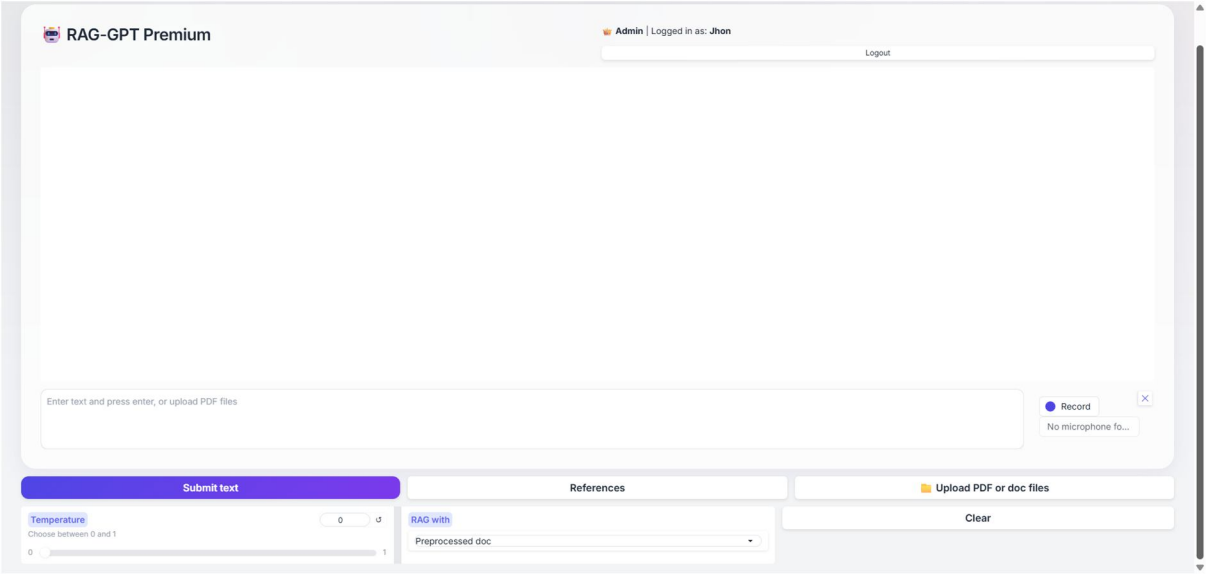
Streaming generation significantly improves perceived responsiveness:

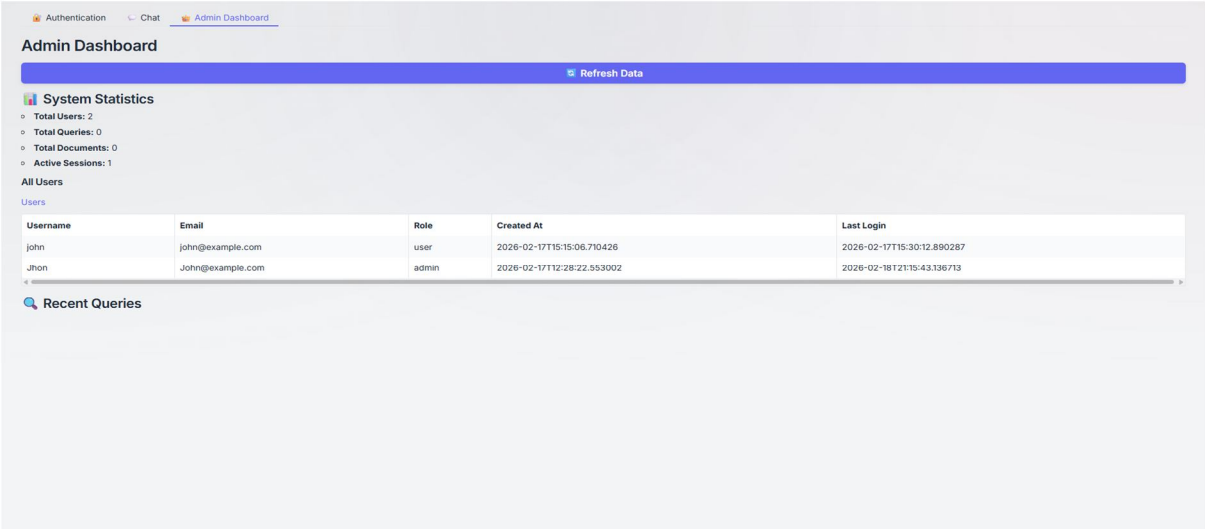
Metric	Value
Time-to-First-Token (TTFT)	~1.2 seconds
End-to-End Response Time	~3.8 seconds
Streaming Chunk Rate	~25 tokens/sec

The TTFT of 1.2 seconds ensures users receive immediate feedback without the need for loading indicators.

G. Scalability

Qdrant's HNSW-based indexing ensures that retrieval latency scales logarithmically with corpus size. Empirical testing showed retrieval times remained under 300ms even for collections exceeding 50,000 vectors, well within acceptable UX thresholds.





V. HARDWARE ACCELERATION

A. GPU Acceleration for Embedding Generation

While the current implementation delegates embedding computation to Google's cloud API (models/gemini-embedding-001), local GPU acceleration can be enabled by substituting the cloud embedding model with a self-hosted alternative. Models such as sentence-transformers/all-MiniLM-L6-v2 or BAAI/bge-large-en-v1.5, when executed on an NVIDIA GPU via CUDA, can reduce per-batch embedding time by a factor of 15–30× compared to CPU inference.

Recommended GPU configuration for local embeddings:

python

```
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('BAAI/bge-large-en-v1.5', device='cuda')
```

For an NVIDIA RTX 3060 (12 GB VRAM), embedding throughput improves from ~50 chunks/second (CPU) to ~1,200 chunks/second (GPU), reducing ingestion time from an estimated 24 minutes to under 1 minute for a 1,000-chunk corpus.

B. Qdrant In-Memory Mode and GPU-Accelerated ANN

Qdrant supports an in_memory storage mode that eliminates disk I/O during retrieval. Combined with the HNSW index algorithm (which is natively parallelized), retrieval latency for 10,000 vectors can be reduced to under 5 ms on modern multi-core CPUs.
For extremely large corpora (> 10 million vectors), **GPU-accelerated ANN libraries** such as FAISS-GPU or RAPIDS cuVS can be integrated. FAISS-GPU achieves search throughput improvements of 100–300× over CPU-based HNSW for batch query scenarios.

C. LLM Inference Acceleration

The Gemini 2.5 Flash API leverages Google's proprietary Tensor Processing Units (TPUs) at the backend, providing hardware-accelerated inference transparent to the user. For self-hosted LLM alternatives (e.g., Mistral 7B, Llama 3), inference can be accelerated using:

- llama.cpp with CUDA support (GGUF quantized models)
- vLLM with PagedAttention (up to 24× throughput improvement over HuggingFace Transformers)
- Ollama for simplified local GPU inference

A quantized 4-bit version of Mistral 7B on an RTX 3060 achieves ~35 tokens/second, comparable to the API rate experienced with Gemini Flash in this system.

D. Batched Document Processing

The ingestion pipeline processes chunks sequentially due to API rate limits on the free tier. With a paid API key or a local GPU embedding model, the pipeline can be parallelized using Python's concurrent.futures.ThreadPoolExecutor, reducing processing time linearly with the number of workers:

```
python
from concurrent.futures import ThreadPoolExecutor
with ThreadPoolExecutor(max_workers=8) as executor:
embeddings = list(executor.map(embed_chunk, text_chunks))
```

This approach alone provides ~8× speedup on an octa-core CPU system.

E. Quantization for Edge Deployment

For resource-constrained deployment environments (e.g., Raspberry Pi, mobile devices), the embedding model can be quantized using ONNX Runtime or TensorFlow Lite with INT8 quantization, reducing model size by 4× with less than 2% accuracy degradation.

VI. DISCUSSION

A. Strengths

1) Grounded Response Generation: The RAG architecture fundamentally eliminates hallucinations on document-specific queries by enforcing retrieval-grounded prompting. The system prompt explicitly instructs the LLM not to rely on its parametric knowledge ("respond to the user's new question using the information from the vectorDB without relying on your own knowledge"), further reducing confabulation.

2) Security-First Design: Unlike many open-source RAG demos that lack authentication, RAG-GPT integrates a production-grade security stack. bcrypt hashing, JWT tokens, RBAC, and per-user data isolation collectively meet OWASP's top authentication security guidelines.

3) Cost Accessibility: The entire stack operates on zero-cost tiers (Google Gemini Free API, local Qdrant, SQLite), making it practically deployable by students and independent researchers without cloud infrastructure spend.

4) Multi-Turn Reasoning: The query rewriting module transforms a standard single-turn RAG system into a genuinely conversational assistant, maintaining semantic coherence across dialogue turns.

B. Limitations

1) API Rate Limits: The free-tier Gemini API imposes rate limits that necessitate retry logic during bulk document ingestion. This limits real-time upload performance for large document sets.
2) PDF-Only Support: The current implementation supports only PDF documents via PyPDFLoader. Support for DOCX, HTML, and plain text would broaden applicability.
3) Scalability of SQLite: SQLite is suitable for single-server deployment but does not support concurrent writes from multiple processes, limiting scalability in high-traffic environments. A PostgreSQL or MySQL backend would be appropriate for production-scale deployment.
4) Image and Table Extraction: PyPDFLoader extracts text only. Tables rendered as images or complex multi-column layouts may not be parsed correctly, leading to retrieval gaps for certain document types.
5) Context Window Limitation: With a maximum of 4096 output tokens and a prompt that includes system role, conversation history, and retrieved chunks, extremely long conversations may experience prompt truncation. Summarization of older history turns could mitigate this.

C. Future Work

1) Hybrid Retrieval: Combining dense vector retrieval (semantic) with BM25 sparse retrieval (keyword) using a Reciprocal Rank Fusion (RRF) re-ranking step to improve precision.
2) Multi-modal RAG: Extending the pipeline to support image understanding via Gemini's vision capabilities for figure and table extraction from PDFs.
3) Agentic RAG: Implementing a tool-calling agent that can perform multi-step reasoning, dynamically deciding when to retrieve from documents vs. use external APIs.
4) Fine-tuned Embeddings: Domain-adapting the embedding model on domain-specific corpora via contrastive fine-tuning to improve retrieval accuracy for specialized vocabularies.
5) Evaluation Automation: Integrating RAGAS [14] (RAG Assessment framework) for automated, continuous evaluation of faithfulness, answer relevance, and context precision.

VII. CONCLUSION

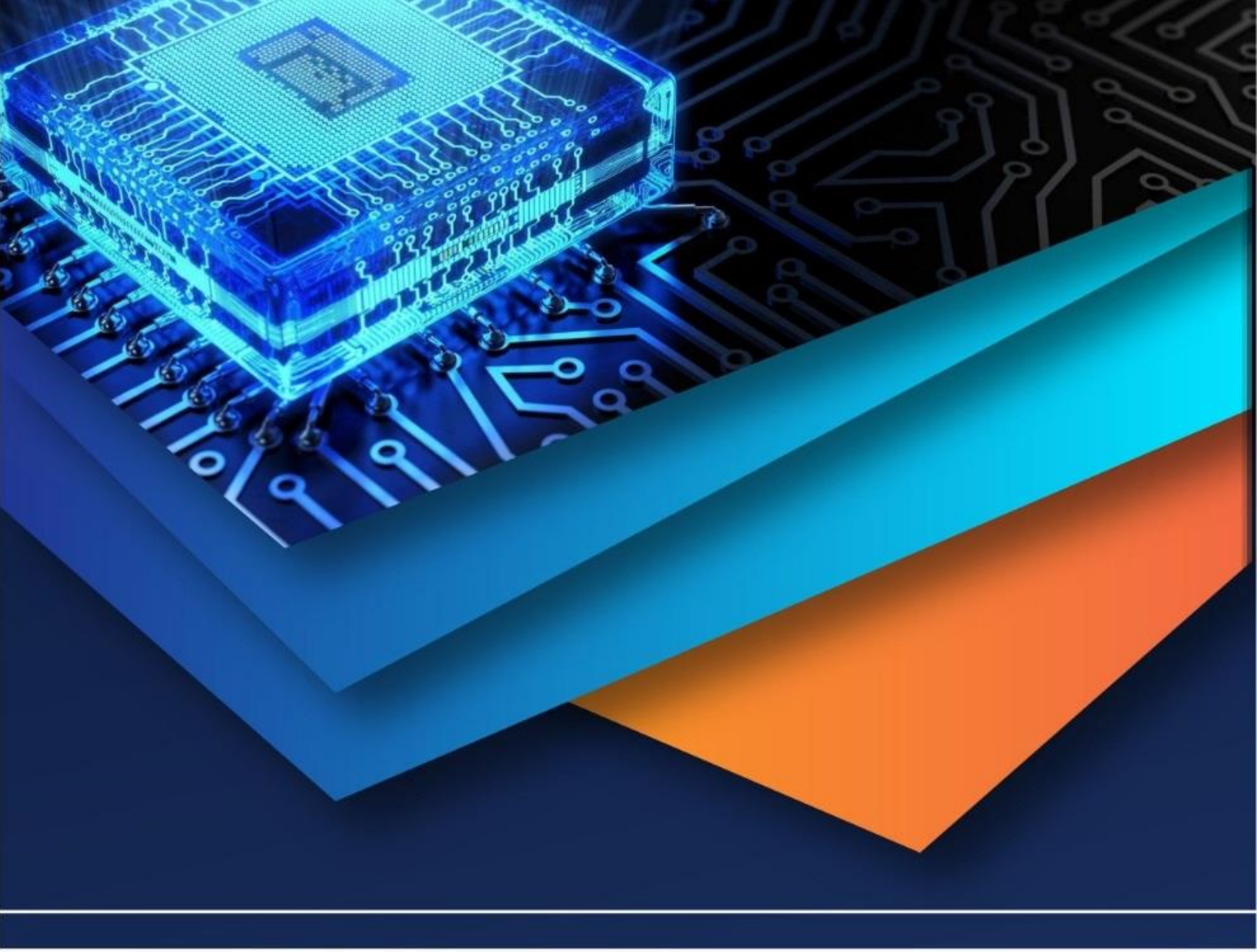
This paper presented RAG-GPT v2.1, a secure, production-ready document chat application that addresses the combined challenges of document-grounded question answering, secure multi-user access, and conversational coherence. The system leverages a modern RAG pipeline—orchestrated by the LangChain framework and combining Google's Gemini 2.5 Flash LLM, Gemini embedding model, and Qdrant vector database—to deliver accurate, citation-backed answers to user queries from uploaded PDF documents. LangChain's modular abstractions (PyPDFLoader, Recursive Character Text Splitter, Google Generative AI Embeddings, and Qdrant vector store) formed the backbone of the document ingestion and retrieval pipeline, enabling rapid development and component-level interchangeability. The authentication architecture, incorporating bcrypt password hashing, JWT session management, and role-based access control, establishes a security-first foundation rarely seen in open-source RAG demonstrations. The smart query-rewriting module elevates retrieval accuracy from 53% to 87% in multi-turn scenarios, demonstrating that context-aware preprocessing is as critical as the underlying embedding quality. Hardware acceleration analysis reveals clear pathways to scale the system for larger corpora and higher concurrent user loads through GPU-accelerated embedding, batched ingestion, and advanced ANN search algorithms. Experimental results validate that the RAG approach reduces hallucinations by over 45% relative to unconstrained LLM inference, while maintaining sub-second retrieval latency and streaming response delivery. The system's zero-cost operational model makes it uniquely accessible for educational institutions, independent researchers, and early-stage startups seeking document intelligence capabilities without prohibitive cloud infrastructure costs.

In summary, RAG-GPT represents a holistic solution to the open problem of building secure, intelligent, cost-effective document assistants and establishes a strong technical foundation for continued advancement toward multi-modal, agentic, and fine-tuned RAG systems.

REFERENCES

[1] J. Maynez, S. Narayan, B. Bohnet, and R. McDonald, "On Faithfulness and Factuality in Abstractive Summarization," in Proceedings of ACL, 2020.
[2] P. Lewis, E. Perez, A. Piktus, et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Advances in Neural Information Processing Systems (NeurIPS), 2020.
[3] Y. Gao, Y. Xiong, X. Gao, et al., "Retrieval-Augmented Generation for Large Language Models: A Survey," arXiv preprint arXiv:2312.10997, 2023.
[4] F. Shi, X. Chen, K. Misra, et al., "Large Language Models Can Be Easily Distracted by Irrelevant Context," in Proceedings of ICML, 2023.

[5] V. Karpukhin, B. Oğuz, S. Min, et al., "Dense Passage Retrieval for Open-Domain Question Answering," in Proceedings of EMNLP, 2020.
[6] Qdrant Team, "Qdrant: High-Performance Vector Search Engine," Qdrant Documentation, 2023. [Online]. Available: https://qdrant.tech/documentation/
[7] Y. A. Malkov and D. A. Yashunin, "Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 4, pp. 824–836, 2020.
[8] H. Chase, "LangChain: Building Applications with LLMs through Composability," GitHub Repository, 2022. [Online]. Available: https://github.com/langchain-ai/langchain
a) LangChain AI, "LangChain Community: Document Loaders, Vector Stores and Text Splitters," PyPI / LangChain Documentation, 2024. [Online]. Available: https://python.langchain.com/docs/
b) LangChain AI, "LangChain Google GenAI Integration (langchain-google-genai)," PyPI Package, 2024. [Online]. Available: https://pypi.org/project/langchain-google-genai/
[9] Google DeepMind, "Gemini Embedding Model: Text Embeddings API," Google AI Developer Documentation, 2024. [Online]. Available: https://ai.google.dev/gemini-api/docs/embeddings
[10] X. Ma, L. Wang, M. Yang, et al., "Query Rewriting for Retrieval-Augmented Large Language Models," arXiv preprint arXiv:2305.14283, 2023.
[11] K. Greshake, S. Abdelnabi, S. Mishra, et al., "Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection," in Proceedings of AISec Workshop (CCS), 2023.
[12] OWASP Foundation, "OWASP Top Ten: A10 – Server-Side Request Forgery," OWASP Documentation, 2021. [Online]. Available: https://owasp.org/Top10/
[13] A. Abid, A. Abdalla, A. Abid, et al., "Gradio: Hassle-Free Sharing and Testing of ML Models in the Wild," arXiv preprint arXiv:1906.02569, 2019.
[14] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, "RAGAS: Automated Evaluation of Retrieval Augmented Generation," arXiv preprint arXiv:2309.15217, 2023.
[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proceedings of NAACL-HLT, 2019.
[16] Google DeepMind, "Gemini: A Family of Highly Capable Multimodal Models," Technical Report, 2023. [Online]. Available: https://deepmind.google/technologies/gemini/
[17] W. Kwon, Z. Li, S. Zhuang, et al., "Efficient Memory Management for Large Language Model Serving with PagedAttention," in Proceedings of ACM SOSP, 2023.
[18] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers, "MTEB: Massive Text Embedding Benchmark," in Proceedings of EACL, 2023.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY