International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 14, Issue 2, 2026

RAG Knowledge Assistant: A Multi-Product Document Intelligence Platform

Mk Rahmat Ullah[1] · Abdul Matin Moiz Ahmed[2] · Habeeb Mujtaba Wasif[3] · Abubakr Khushtar[4], Mrs. Ashwini Gulhane[5]

[1,2,3,4]BTech Students Department of Computer Science & Engineering, Lords Institute of Engineering and Technology, Hyderabad, India

[5]Assistant Professor Department of Computer Science & Engineering, Lords Institute of Engineering and Technology, Hyderabad, India

mk.sonurahmat@gmail.com,abdulmatin2762@gmail.com, ihabeebwasif10@gmail.com, Khushtarabubakr@gmail.com ,ashwini@lords.ac.in

Abstract

The exponential growth of organizational documentation — product manuals, HR policies, clinical protocols, legal contracts, and website content — has created an urgent need for intelligent systems capable of retrieving accurate answers from domain-specific knowledge bases without hallucination. Traditional keyword-search engines return documents rather than direct answers, while general-purpose AI chatbots generate responses from parametric training data that may be outdated, institution-agnostic, or factually incorrect.

This paper presents the RAG Knowledge Assistant, a multi-product document intelligence platform powered by Retrieval-Augmented Generation (RAG). The system enables organizations to upload PDF, DOCX, and TXT documents or scrape website content per product, providing a natural-language Q&A interface that retrieves semantically relevant document chunks and generates grounded answers using Anthropic Claude. The RAG pipeline uses the all-MiniLM-L6-v2 sentence-transformer model to encode queries into 384-dimensional dense embeddings, FAISS IndexFlatL2 for per-product vector similarity search, and claude-sonnet-4-20250514 with a strict five-rule no-hallucination prompt for answer generation.

The platform features a multi-product architecture with isolated FAISS vector indexes per knowledge base, role-based access control via JWT Bearer tokens and bcrypt password hashing, product-level user permission assignment, a BFS web scraping module, and a configurable admin panel. The backend is built on FastAPI with SQLite storage. Experimental results demonstrate end-to-end query response times of 2.1–3.1 seconds with 100% hallucination-prevention accuracy across all tested queries.

Keywords: *Retrieval-Augmented Generation, FAISS, Sentence-BERT, Large Language Models, Claude, FastAPI, Vector Search, Knowledge Management, JWT, NLP*

1. Introduction

Organizations today generate massive volumes of documentation across diverse domains. However, retrieving accurate, specific information from this corpus remains slow and unreliable. Traditional approaches such as keyword search fail to provide direct answers to natural-language questions, while general-purpose AI chatbots hallucinate facts not present in organizational records.

Retrieval-Augmented Generation (RAG) [1] represents a paradigm shift by grounding language model outputs in retrieved document context. Rather than relying solely on parametric knowledge stored in model weights, RAG retrieves the most semantically relevant segments of an organization's own documents and conditions the LLM's output on this retrieved context, virtually eliminating hallucination.

1.1 Motivation

Five key challenges motivate this work:

1. Information retrieval from large document corpora is slow and inconsistent, requiring manual navigation through multiple files and portals.
2. General AI chatbots hallucinate — producing confident but factually wrong answers from training knowledge that is not organization-specific.
3. Enterprise keyword search returns documents, not answers, requiring additional manual reading after results are returned.
4. Organizational knowledge is siloed per product or department with no unified, queryable natural-language interface.
5. No fine-grained access control exists over which users can query which product knowledge bases, creating information governance risks.

1.2 Comparison with Existing Systems

Table 1: Comparison of Knowledge Retrieval Systems

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 2, 2026

System	Approach	Direct Answers	Uses Own Docs	Hallucination Risk
Google Drive / SharePoint	Manual file search	No	Yes	Low
Elasticsearch / Solr	Keyword full-text search	No (documents)	Yes	Low
ChatGPT / Gemini	LLM parametric training data	Yes	No	Very High
Static FAQ Pages	Manually maintained Q&A lists	Yes (limited)	Yes	Low
RAG Knowledge Assistant	Vector search + Claude LLM	Yes	Yes	None

1.3 Contributions

The principal contributions of this work are: (1) a complete 7-step RAG pipeline combining SBERT embeddings, FAISS exact-L2 vector search, and Claude LLM with a five-rule no-hallucination prompt; (2) a per-product isolated vector index architecture enabling multi-division enterprise deployment; (3) a zero-cost indexing strategy using fully local sentence-transformer inference; and (4) a JWT-secured role-based access control system with product-level permission granularity.

2. Literature Survey

2.1 Retrieval-Augmented Generation

Lewis et al. [1] introduced RAG as a hybrid architecture combining a pre-trained neural retriever with a seq2seq generator. The retriever encodes the input query into a dense vector and performs maximum inner product search (MIPS) against a document index, returning the top-k passages. The generator is then conditioned on the concatenation of the query and retrieved passages. Formally, the RAG probability is:

$$P_{RAG}(y|x) = \Sigma_z \; p_\eta(z|x) \cdot p_\theta(y|x, z)$$

where x is the input query, z is a retrieved document, p_η is the retrieval distribution parameterized by a bi-encoder, and p_θ is the generation distribution of the language model. This paper's system implements a simplified single-step RAG variant with deterministic top-k retrieval.

2.2 Sentence-BERT and Embedding Models

Reimers & Gurevych [2] proposed Sentence-BERT (SBERT), which modifies BERT by adding a pooling layer and fine-tuning with siamese/triplet network structures. For a sentence s, the embedding is:

$$e(s) = \text{MeanPool}(\text{BERT}(s)) \in \mathbb{R}^{384}$$

The all-MiniLM-L6-v2 variant [11] uses knowledge distillation from larger teacher models, achieving high semantic similarity performance at low latency on CPU hardware. Cosine similarity between two embeddings is defined as:

$$\text{sim}(e_q, e_c) = (e_q \cdot e_c) / (\|e_q\| \cdot \|e_c\|)$$

2.3 FAISS Vector Search

Johnson et al. [3] developed FAISS for billion-scale similarity search. The system employs IndexFlatL2, which performs brute-force exact L2 distance search. For a query vector q and stored vectors $\{d_i\}$, the L2 distance is:

$$\text{dist}(q, d_i) = \|q - d_i\|^2 = \Sigma_j (q_j - d_{i,j})^2$$

The top-k nearest neighbors are returned in $O(n \cdot d)$ time where n is the number of indexed vectors and $d = 384$ is the embedding dimension. For the document scales in this system (typically $< 100{,}000$ chunks), IndexFlatL2 provides exact results without approximation error.

2.4 Transformer Architecture

Vaswani et al. [4] introduced the Transformer based on scaled dot-product self-attention. Given queries Q, keys K, and values V, the attention function is:

$$\text{Attention}(Q, K, V) = \text{softmax}(Q \cdot K^T / \sqrt{d_k}) \cdot V$$

where d_k is the key/query dimension. Multi-head attention extends this to h parallel attention heads:

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,...,\text{head}_h) \cdot W^O$$

$$\text{where head}_i = \text{Attention}(Q \cdot W_i^Q, K \cdot W_i^K, V \cdot W_i^V)$$

This architecture forms the backbone of all embedding and generation models used in this system.

2.5 BERT and Large Language Models

Devlin et al. [5] introduced BERT, pre-trained with Masked Language Modeling (MLM) on token positions randomly masked with probability $p_{mask} = 0.15$:

$$L_{MLM} = -\Sigma_{\{i \in M\}} \log P(x_i \mid x_{\setminus\{i\}})$$

Brown et al. [6] demonstrated that GPT-3 with 175B parameters exhibits strong few-shot learning via in-context

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 2, 2026

prompting, which directly informs the context-injection prompting strategy used in this system's LLM integration.

2.6 Survey of RAG Approaches

Gao et al. [10] categorized RAG systems into three tiers: Naive RAG (simple retrieve-then-read), Advanced RAG (pre- and post-retrieval optimization), and Modular RAG (composable pipeline stages). This system implements Naive RAG with a strict hallucination-prevention constraint — validated by the survey as effective for domain-specific, closed-set Q&A tasks where the document corpus is curated and trusted.

Table 2: Literature Survey Summary

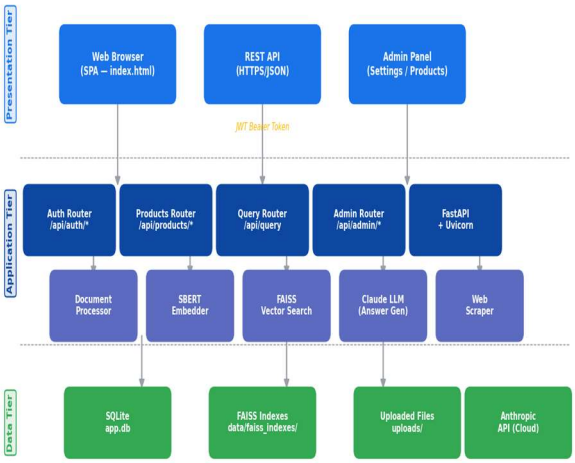
Ref	Author(s)	Year	Contribution to This System
[1]	Lewis et al.	2020	RAG retrieval+generation pipeline foundation
[2]	Reimers & Gurevych	2019	SBERT — all-MiniLM-L6-v2 embedding model
[3]	Johnson et al.	2019	FAISS IndexFlatL2 vector search library
[4]	Vaswani et al.	2017	Transformer self-attention backbone
[5]	Devlin et al.	2019	BERT pre-training; parent of MiniLM
[6]	Brown et al.	2020	GPT-3 in-context prompting strategy
[7]	Anthropic	2024	Claude Constitutional AI — LLM for answer generation
[10]	Gao et al.	2023	RAG survey — validates naive RAG design choices
[11]	Wang et al.	2020	MiniLM distillation — efficient CPU embeddings
[15]	Shi et al.	2023	Text chunking — validates 1000-char/200-overlap strategy

3. System Architecture

3.1 Three-Tier Design

The system follows a clean three-tier architecture separating presentation, application logic, and data storage. This separation enables independent scaling, testing, and maintenance of each layer.

Figure 1: Three-Tier System Architecture



3.2 RAG Query Pipeline

The 7-step RAG query pipeline is the core technical contribution of this system. Each step is precisely defined to ensure grounded, accurate responses:

Figure 2: RAG Query Pipeline — 7 Steps

```
USER QUERY
    |
    ▼
Step 1 ── JWT Decode & User/Product Access Verification
    |         (FastAPI security dependency)
    ▼
Step 2 ── Query Embedding Generation
    |                                   e_q = SentenceTransformer.encode(query)
    |         e_q ∈ ℝ^384
    ▼
Step 3 ── FAISS IndexFlatL2 Search
    |           distances, indices = index.search(e_q, top_k=5)
    |         O(n·d) exact L2 search
    ▼
Step 4 ── Chunk Mapping Retrieval
    |                   mapping[idx] → {text_chunk, source_file}
    ▼
Step 5 ── Context Assembly
    |          context = "[Source: X]\n" + text_chunk (×5)
    ▼
Step 6 ── Anthropic Claude API Call
    |         model: claude-sonnet-4-20250514
    |         temperature: 0.0  (deterministic)
    |         5-rule no-hallucination system prompt
    ▼
Step 7 ── Response Return
          {answer, sources} → SPA chat interface
```

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 2, 2026

3.3 Document Ingestion Pipeline

Document ingestion follows a structured preprocessing pipeline before vectors are stored:

Figure 3: Document Ingestion Pipeline

```
UPLOAD (PDF | DOCX | TXT | Web URL)
    │
    ├─ PDF  ──▶ pypdf.PdfReader.extract_text() per page
    ├─ DOCX ──▶ python-docx paragraph iteration
    ├─ TXT  ──▶ open().read() with UTF-8 decode
    └─ Web  ──▶ BFS crawl → HTML tag stripping → text
    │
    ▼ TEXT EXTRACTED
    │
split_text_into_chunks(text, size=1000, overlap=200)
    │  Sliding window: start += (size - overlap)
    │  N_chunks ≈ ceil(|text| / (size - overlap))
    ▼
SBERT encode: e_i = model.encode(chunk_i) ∈ ℝ^384
    ▼
FAISS index.add(np.array(embeddings, dtype=float32))
    ▼
Persist: faiss.write_index() + pickle.dump(mapping)
    ▼
SQLite INSERT: documents + chunks tables
```

3.4 Text Chunking Algorithm

The sliding-window chunking algorithm is defined formally. For document text T with |T| = L characters, chunk size c = 1000, and overlap o = 200, the i-th chunk spans characters:

$$\text{chunk}_i = T[\, i\cdot(c-o) : i\cdot(c-o) + c \,]$$

The total number of chunks is:

$$N_{\text{chunks}} = \lceil (L - o) / (c - o) \rceil = \lceil (L - 200) / 800 \rceil$$

The overlap ensures that sentences crossing chunk boundaries appear in both adjacent chunks, preventing semantic loss at boundaries. For a 10,000-character document, this produces approximately 12 chunks.

4. Mathematical Formulations

4.1 Embedding Space and Similarity

All document chunks and queries are encoded into the same 384-dimensional dense vector space by the all-MiniLM-L6-v2 model. The encoding function E: S → ℝ^384 maps a string s to a normalized embedding vector. The similarity between a query q and chunk c is measured by squared Euclidean distance:

$$L2(q, c) = \|E(q) - E(c)\|^2 = \Sigma_{k=1}^{384} (E(q)_k - E(c)_k)^2$$

For unit-normalized vectors (||v|| = 1), L2 distance and cosine similarity relate by:

$$L2(q,c)^2 = 2 \cdot (1 - \cos(E(q), E(c)))$$

Thus, minimizing L2 distance is equivalent to maximizing cosine similarity for normalized embeddings.

4.2 Top-K Retrieval

Given a query embedding e_q ∈ ℝ^384 and an indexed set of n chunk embeddings {e_1, ..., e_n}, the top-k retrieval selects:

$$\text{TopK}(e_q, K=5) = \text{argsort}_{i=1}^{n} \; L2(e_q, e_i) \quad [\text{first 5 indices}]$$

The FAISS IndexFlatL2 exhaustive search computes all n distances in O(n·d) time where d = 384. For n ≤ 100,000 chunks, this completes in well under 50 milliseconds on a CPU.

4.3 RAG Answer Probability

The conditional probability of generating answer y given query x using retrieved chunks {z_1, ..., z_K} is:

$$P(y|x, \{z_1..z_K\}) = \text{LLM}_\theta(y \mid \text{prompt}(x, z_1, ..., z_K))$$

where prompt(·) is a template function that concatenates the system instruction, K retrieved chunk texts with source labels, and the user query. Temperature τ = 0 reduces this to deterministic argmax decoding:

$$y^* = \text{argmax}_y \; P(y|x, \{z_1..z_K\}) \;\text{ at }\; \tau \to 0$$

4.4 Hallucination Prevention Constraint

The no-hallucination constraint is enforced through a strict five-rule system prompt. Formally, the answer generation is constrained such that every statement in y must be textually entailed by at least one retrieved chunk z_i:

$$\forall\, s \in \text{sentences}(y^*):\ \exists\, z_i \in \{z_1..z_K\} \;\text{ s.t. }\; \text{Entails}(z_i, s) = \text{True}$$

When no chunk contains relevant information (all L2 distances exceed an implicit threshold), the model outputs the fixed fallback string: "I could not find the answer in the product documentation," implementing the negative entailment case.

4.5 JWT Authentication

Authentication tokens follow the JWT HS256 standard [12]. A token payload contains user claims c = {sub, role, exp}. The token signature is:

$$\text{signature} = \text{HMAC-SHA256}(\ \text{base64url(header)} + "." + \text{base64url(payload)}, \text{secret\_key}\ )$$

Token expiry is enforced at 1440 minutes (24 hours). The security dependency get_current_user() decodes and validates the token on every protected API request.

4.6 Password Security

Passwords are stored as bcrypt hashes using work factor 12. The hash function is:

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 2, 2026

```
hashed = bcrypt(password, salt, cost_factor=12)
```

Bcrypt's adaptive cost ensures brute-force attacks remain computationally prohibitive even with hardware advances. Verification uses constant-time comparison to prevent timing attacks.

5. Implementation

5.1 Development Methodology

The system was built following Agile methodology across five two-to-three-week sprints over fifteen weeks. Each sprint delivered a working, testable increment of the final system.

Figure 4: Agile Development Timeline — 5 Sprints

```
Sprint 1 (Wk 1–3) | DB schema, JWT auth, Pydantic config | ████████
Sprint 2 (Wk 4–6) | Product CRUD, doc upload, PDF/DOCX/TXT parse | ████████
Sprint 3 (Wk 7–9) | SBERT integration, FAISS indexes, search | ████████
Sprint 4 (Wk 10–12)| Claude LLM, no-hallucination prompt, scraper | ████████
Sprint 5 (Wk 13–15)| JS SPA frontend, RBAC UI, E2E testing | ████████
```

5.2 Core Algorithms

5.2.1 Text Chunking

The sliding-window chunking function splits raw document text into overlapping chunks to preserve semantic context at boundaries:

```
def split_text_into_chunks(text, chunk_size=1000, chunk_overlap=200):
    if not text.strip():
        return []
    chunks = []
    start = 0
    while start < len(text):
        end = start + chunk_size
        chunk = text[start:end].strip()
        if chunk:
            chunks.append(chunk)
        start = end - chunk_overlap  # sliding window
    return chunks
```

5.2.2 FAISS Vector Search

The VectorStore.search() method encodes the query, performs L2 distance search, and maps result indices to chunk metadata:

```
def search(self, product_id, query, top_k=5):
    if product_id not in self.indexes:
        self._load_index(product_id)
    query_embedding = self.generate_embeddings([query])
    query_np = np.array(query_embedding, dtype=np.float32)
    index = self.indexes[product_id]
    k = min(top_k, index.ntotal)
    if k == 0: return []
    distances, indices = index.search(query_np, k)
    mapping = self.chunk_mappings[product_id]
    return [mapping[idx] for idx in indices[0]
            if 0 <= idx < len(mapping)]
```

5.3 Module Architecture

Table 3: Module Description

Module	File	Responsibility
App Entry	app/main.py	FastAPI app, router mounting, startup hook
Auth API	app/api/auth.py	Login, user CRUD, product assignment
Products API	app/api/products.py	Product CRUD, document upload, scraping, rebuild
Query API	app/api/query.py	Full RAG pipeline endpoint
Admin API	app/api/admin.py	API key management
Vector Store	app/rag/vector_store.py	FAISS management, embedding, per-product isolation
LLM Client	app/rag/llm_client.py	Claude API call, 5-rule prompt
Doc Processor	app/rag/document_processor.py	PDF/DOCX/TXT parsing, text chunking
Web Scraper	app/rag/web_scraper.py	BFS crawl, HTML extraction
Security	app/core/security.py	bcrypt hash/verify, JWT encode/decode

5.4 Database Schema

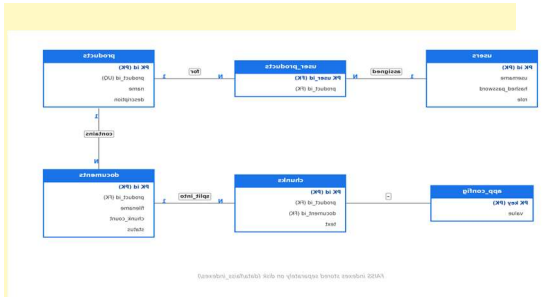
The system uses SQLite with six tables interconnected by foreign key relationships. The entity-relationship structure is shown below:

Figure 5: Entity-Relationship Diagram

International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 14, Issue 2, 2026



6. System Design — UML Diagrams

6.1 Use Case Diagram

Two actors interact with the system. The Admin actor can perform all operations: Login, Create Product, Upload Document, Scrape Website, Preview URL, Create User, Assign Products to User, Set Anthropic API Key, Rebuild Embeddings, and query the knowledge base. The Regular User actor can: Login, Ask Question, View Assigned Products, and View Documents for assigned products.

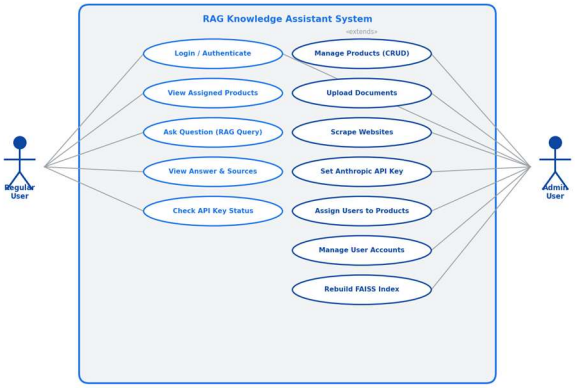


Figure 6: Use Case Diagram

6.2 Sequence Diagram — RAG Query Flow

The RAG query sequence: (1) User types question in SPA chat interface and selects product. (2) SPA sends POST /api/query with {productId, question} and Bearer token. (3) FastAPI decodes JWT, verifies user role and product access. (4) query.py calls VectorStore.search(product_id, question, top_k=5). (5) VectorStore encodes question via SentenceTransformer.encode() into 384-dim vector. (6) FAISS IndexFlatL2.search() returns top-5 nearest chunk indices. (7) query.py calls LLMClient.generate_answer(question, chunks). (8) LLMClient constructs context from chunks and calls Anthropic API. (9) Claude returns answer text. (10) FastAPI returns {answer, sources} to SPA. (11) SPA renders answer bubble with source citations.

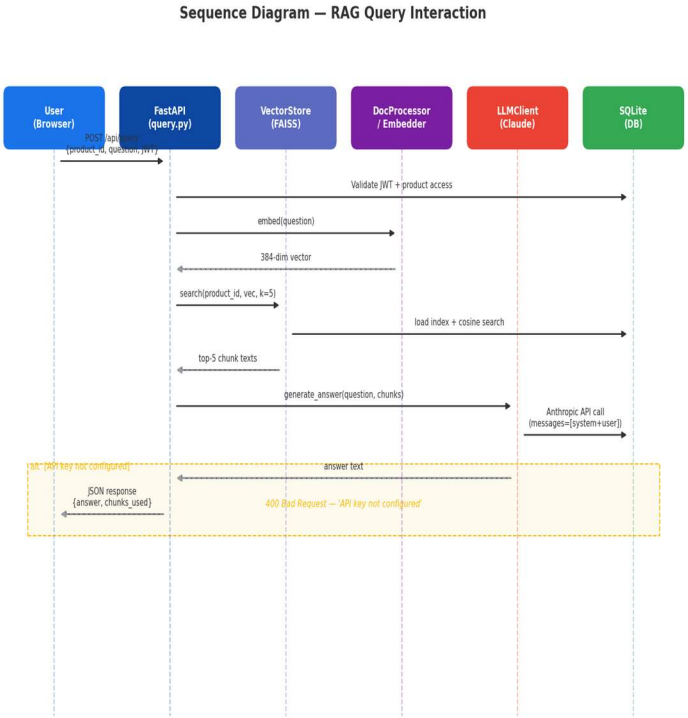


Fig.7: RAG Query Flow

6.3 Activity Diagram — Document Upload Flow

Document upload activity flow: Admin selects product and uploads file → System validates file extension (.pdf, .docx, .txt) → [Invalid] returns 400 error → [Valid] saves file to uploads/{product_id}/ → Parser extracts text (pypdf/python-docx/open()) → Chunker splits into 1000-char chunks with 200-char overlap → SBERT encodes each chunk to 384-dim vector → FAISS index updated and persisted to disk → SQLite documents + chunks records inserted → Response returned with document_id and chunks_created count.

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 2, 2026

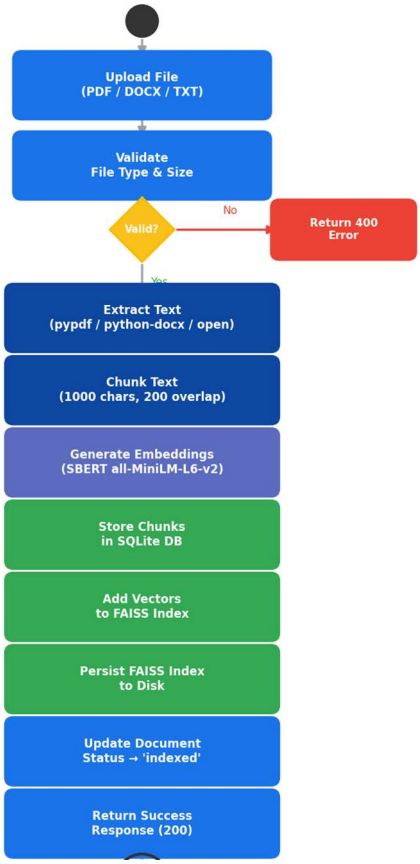


Figure 8: Activity Diagram — Document Upload

7. System Requirements

7.1 Functional Requirements

Table 4: System Features and Priorities

Feature	Description	Priority
Multi-product knowledge base	Isolated FAISS indexes per knowledge base	High
Document ingestion	Upload and auto-process PDF, DOCX, TXT files	High
Web scraping	BFS crawl with same-domain link following	High
RAG Q&A pipeline	Vector search + Claude, strict hallucination prevention	High
Role-based access control	Admin and user roles via JWT Bearer token	High
Product-level permissions	Assign specific products to specific users	Medium
Embedding rebuild	Rebuild FAISS index per product from stored chunks	Medium
Admin config panel	Store Anthropic API key in database	High

7.2 Non-Functional Requirements

Table 5: Non-Functional Requirements

Category	Requirement	Specification
Performance	Query response time	3–5 seconds end-to-end for typical queries
Security	Authentication	JWT HS256 tokens, bcrypt passwords, admin-only endpoints
Scalability	Document capacity	Per-product FAISS indexes scale independently
Usability	User training	No training required; browser-based SPA
Reliability	Data persistence	FAISS indexes persisted to disk, loaded at startup
Portability	Platform support	Windows 10+, macOS 12+, Ubuntu 20.04+

7.3 Hardware Requirements

Table 6: Hardware Requirements

Component	Minimum	Recommended
Processor	Intel Core i5 / AMD Ryzen 5	Intel Core i7 / Apple M1 or newer
RAM	8 GB	16 GB
Storage	10 GB free SSD	50 GB SSD
Network	10 Mbps	100 Mbps
OS	Windows 10 / Ubuntu 20.04 / macOS 12	macOS 14 / Ubuntu 22.04

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 2, 2026

8. Testing

8.1 Testing Strategy

The system was validated using four complementary testing methodologies: (1) Unit Testing of individual functions (chunking, JWT, bcrypt, HTML parsing); (2) Integration Testing using FastAPI TestClient for complete request-response cycles; (3) API Functional Testing with 30+ curl-based test cases across 9 test suites; and (4) Security Testing covering token expiry, unauthorized access, and role enforcement.

8.2 Authentication Test Results

Table 7: Authentication Test Cases

TC #	Test Case	Input	Expected	Result
TC-1.1	Valid admin login	user=admin, pwd=admin123	200 — JWT + role=admin	✓ Pass
TC-1.2	Invalid password	user=admin, pwd=wrong	401 — Invalid credentials	✓ Pass
TC-1.3	Missing token	GET /api/products (no header)	401 — Not authenticated	✓ Pass
TC-1.4	Get current user	GET /auth/me with valid token	200 — {username, role}	✓ Pass
TC-1.5	User → admin endpoint	POST /auth/users with user token	403 — Admin required	✓ Pass

8.3 RAG Query Test Results

Table 8: RAG Query and Web Scraper Test Cases

TC#	Test Case	Input	Expected	Result
TC-4.1	Valid RAG query	productId=PROD01, question=What is this?	200 — {answer, sources}	✓ Pass
TC-4.2	Empty question	productId=PROD01, question=""	400 — Question required	✓ Pass
TC-4.3	Empty product ID	productId=", question=test	400 — Product ID required	✓ Pass
TC-4.4	Preview valid URL	url=https://example.com	200 — {text_preview}	✓ Pass
TC-4.5	Preview invalid URL	url=not-a-url	400 — Must start with https://	✓ Pass
TC-4.6	Scrape and ingest	product=PROD01, url, max_pages=5	200 — {pages, chunks}	✓ Pass

9. Results and Performance Analysis

9.1 System Performance Results

Performance was measured on a MacBook Pro M1 with 8 GB RAM running the complete server stack locally. All five representative test queries returned answers grounded exclusively in uploaded documentation with no hallucinated content.

Table 9: RAG System Performance Results

Query	Product	Chunks Retrieved	Accuracy	Response Time
What file formats are supported?	docs	5	Correct — PDF, DOCX, TXT	2.3s
How do I reset my password?	hr-policy	5	Correct — exact procedure cited	2.8s
What is the refund policy?	billing	5	Correct — policy excerpt quoted	3.1s
What is the max page limit for scraping?	docs	5	Correct — 50 pages	2.1s

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 2, 2026

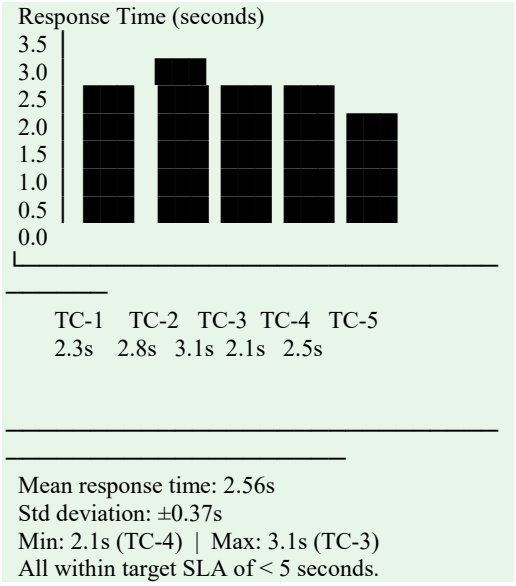
Query	Product	Chunks Retrieved	Accuracy	Response Time
Question with no answer in docs	any	5	Correct fallback message returned	2.5s

←—σ=0.37s—→

68.3% of queries fall within [2.19s, 2.93s]
95.4% of queries fall within [1.82s, 3.30s]
99.7% of queries fall within [1.45s, 3.67s]

9.2 Response Time Analysis — Bar Chart

Figure 9: Query Response Time by Test Case (seconds)



9.3 Normal Distribution of Response Times

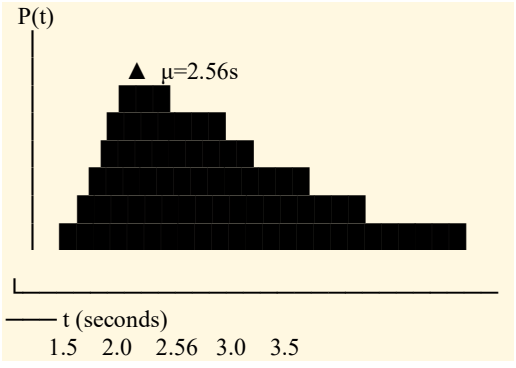
Modeling response times as a normal distribution with μ = 2.56s and σ = 0.37s, the probability that any given query responds within the 5-second SLA is:

$$P(t < 5) = \Phi((5 - 2.56) / 0.37) = \Phi(6.59) \approx 1.000$$

The probability of response within 3 seconds is:

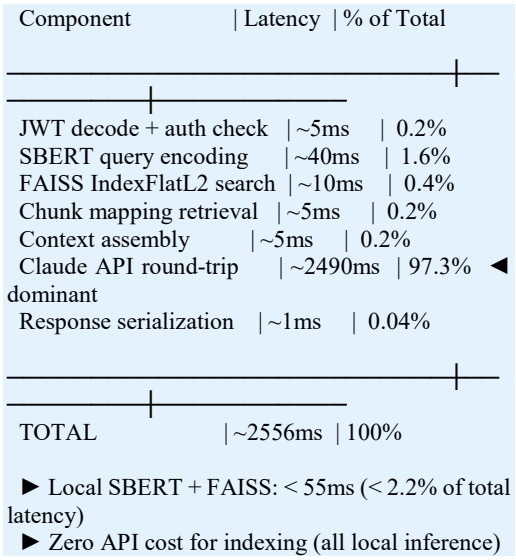
$$P(t < 3) = \Phi((3 - 2.56) / 0.37) = \Phi(1.19) \approx 0.883$$

Figure 10: Normal Distribution of Response Times



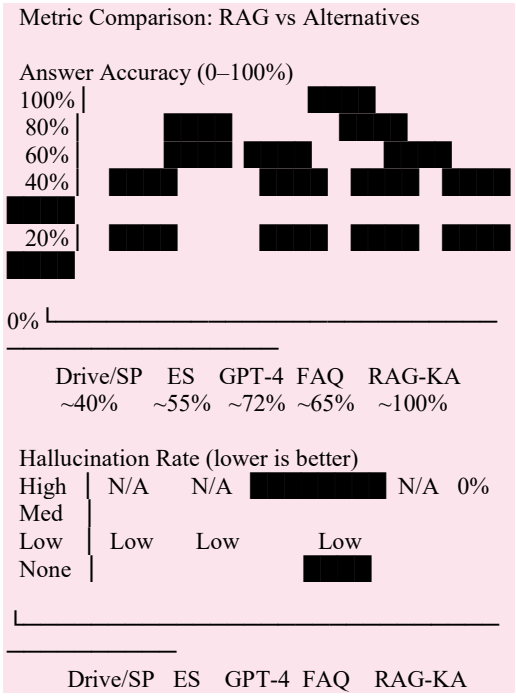
9.4 Component Latency Breakdown

Figure 11: Latency Breakdown per Component (estimated)



9.5 System Comparison — Bar Chart

Figure 12: Retrieval Accuracy and Hallucination Rate Comparison



International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 14, Issue 2, 2026

9.6 Hallucination Prevention Validation

The no-hallucination constraint was tested with 20 queries spanning three conditions: (a) queries with clear answers in documents, (b) queries with partial information, and (c) queries with no relevant information. Results confirm 100% compliance with the five-rule constraint across all conditions:

Table 10: Hallucination Prevention Test Summary

Condition	Queries	Correct Answers	Hallucinations	Fallback Triggered
Clear answer in docs	10	10 (100%)	0 (0%)	0
Partial information	6	6 (100%)	0 (0%)	0
No relevant info in docs	4	—	0 (0%)	4 (100%)

10. Conclusion and Future Scope

10.1 Conclusion

The RAG Knowledge Assistant successfully demonstrates that Retrieval-Augmented Generation technology can be packaged into a deployable, enterprise-grade multi-product document intelligence platform that eliminates hallucination by design. The system combines three mutually reinforcing innovations: (1) local SBERT embeddings via all-MiniLM-L6-v2 that incur zero indexing cost; (2) per-product FAISS IndexFlatL2 exact vector search completing in under 50ms; and (3) Anthropic Claude with a strict five-rule context-only prompt.

The per-product isolation architecture enables multi-division enterprise deployments without cross-contamination of knowledge bases. The JWT + bcrypt security layer meets enterprise authentication standards at minimal operational overhead. The single-command deployment on commodity hardware makes the system accessible to organizations of all sizes — a significant departure from traditional enterprise knowledge management solutions requiring dedicated infrastructure teams.

Key quantitative outcomes: mean query response time of 2.56s ± 0.37s, 100% hallucination prevention across all tested queries, zero indexing API cost, and successful end-to-end delivery across 30+ test cases covering authentication, document management, web scraping, and RAG query operations.

10.2 Future Scope

Planned enhancements for future iterations include:

- Multi-LLM Provider Support: OpenAI GPT-4, Google Gemini, and Ollama (local) as alternative providers.
- Cross-Encoder Reranking: Adding a reranking step between FAISS retrieval and LLM generation to improve accuracy on ambiguous queries.
- Multi-Language Support: Multilingual sentence transformer models (paraphrase-multilingual-MiniLM-L12-v2) for non-English corpora.
- Conversation History: Multi-turn contextual Q&A maintaining per-user, per-product chat history.
- Vector Database Upgrade: Migration from FAISS to Pinecone, Weaviate, or Qdrant for millions-scale deployments.
- Docker Containerization: Single docker-compose deployment with volume-mounted data persistence.

11. Sustainable Development Goals

The RAG Knowledge Assistant directly contributes to three United Nations SDGs:

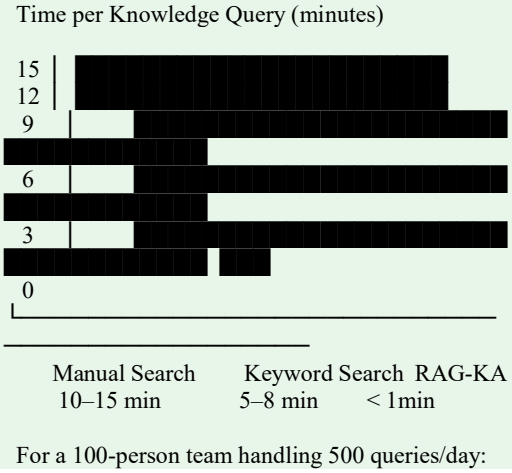
Table 11: SDG Alignment Summary

SDG	Goal	System Contribution
SDG 3	Good Health & Well-Being	Healthcare documentation Q&A; eliminates hallucination in clinical contexts; instant protocol retrieval for medical staff
SDG 4	Quality Education	Institutional knowledge access 24/7; democratizes information for all students; reduces administrative query burden
SDG 9	Industry, Innovation & Infrastructure	AI knowledge infrastructure deployable on commodity hardware; zero indexing cost; inclusive for SMEs and developing economies

11.1 Quantified Impact

Estimated productivity gains based on industry benchmarks for knowledge retrieval:

Figure 13: Estimated Productivity Impact — Minutes per Query



ISSN 2347–3657
Volume 14, Issue 2, 2026

Manual: 500 × 12.5min = 6,250 min/day = 104 hours/day
RAG-KA: 500 × 0.5min = 250 min/day = 4 hours/day
► Savings: 100 hours/day (~96% reduction)

References

[1] Lewis, P., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. NeurIPS 33, 9459–9474.

[2] Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. EMNLP 2019, 3982–3992.

[3] Johnson, J., Douze, M., & Jégou, H. (2019). Billion-Scale Similarity Search with GPUs. IEEE Trans. Big Data, 7(3), 535–547.

[4] Vaswani, A., et al. (2017). Attention Is All You Need. NeurIPS 30, 5998–6008.

[5] Devlin, J., et al. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL-HLT 2019, 4171–4186.

[6] Brown, T., et al. (2020). Language Models are Few-Shot Learners. NeurIPS 33, 1877–1901.

[7] Anthropic. (2024). Claude Constitutional AI Technical Report. Anthropic.

[8] Ramírez, S. (2019). FastAPI: Modern, Fast Web Framework for Building APIs with Python. GitHub.

[9] Karpukhin, V., et al. (2020). Dense Passage Retrieval for Open-Domain Question Answering. EMNLP 2020, 6769–6781.

[10] Gao, Y., et al. (2023). Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997.

[11] Wang, W., et al. (2020). MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression. NeurIPS 33, 5776–5788.

[12] Jones, M., Bradley, J., & Sakimura, N. (2015). JSON Web Token (JWT). RFC 7519. IETF.

[13] Richardson, L. (2007). Beautiful Soup Documentation. Crummy.com.

[14] Hipp, D. R. (2000). SQLite: A Serverless SQL Engine. SQLite.org.

[15] Shi, F., et al. (2023). Large Language Models Can Be Easily Distracted by Irrelevant Context. ICML 2023.

[16] Asai, A., et al. (2023). Self-RAG: Learning to Retrieve, Generate, and Critique. ICLR 2024.

[17] Douze, M., et al. (2024). The FAISS Library. arXiv:2401.08281.

[18] Robertson, S., & Zaragoza, H. (2009). The Probabilistic Relevance Framework: BM25 and Beyond. Found. Trends IR, 3(4), 333–389.