

LLM-INTEGRATED ARCHITECTURE FOR DOCUMENT INTELLIGENCE AND SEMANTIC RETRIEVAL

Pham Minh Tuan¹, Nguyen Cong Cuong¹, Quoc Khanh Trinh¹, Phuc Hao Do², Nguyen Nang Hung Van^{1*}

¹*The University of Danang - University of Science and Technology, Vietnam*

²*Danang Architecture University, Vietnam*

*Corresponding author: nguyenvan@dut.udn.vn

(Received: April 13, 2026; Revised: September 07, 2026; Accepted: September 10, 2026)

DOI: 10.31130/ud-jst.2026.24(9).221E

Abstract - This study presents an Artificial Intelligence-enabled digital transformation system for intelligent enterprise document management. The proposed platform combines Optical Character Recognition, semantic embeddings, vector search, and Retrieval-Augmented Generation to improve document retrieval and knowledge access in organizational environments. Unlike traditional keyword-based systems, it supports context-aware search and evidence-grounded question answering over internal documents. The system is implemented with a practical full-stack architecture using React, FastAPI, and PostgreSQL with pgvector. To improve retrieval quality, we introduce a K-Means-based filtering step that separates highly relevant results from noisy ones before answer generation. Experimental observations show that this hybrid approach provides more focused outputs and better practical precision while preserving strong ranking quality. The results demonstrate strong potential for scalable enterprise knowledge intelligence applications.

Key words – Artificial Intelligence; Document Intelligence; Semantic Retrieval; Retrieval-Augmented Generation; Enterprise Knowledge Management

1. Introduction

Digital transformation has become a strategic imperative for enterprises, universities, and public organizations that manage large volumes of heterogeneous documents. Traditional document workflows are typically fragmented across local files, email attachments, and departmental repositories, resulting in inefficient retrieval, duplicated effort, and weak knowledge reuse. Although many organizations have already digitized storage, most existing systems still rely on keyword indexing and manual metadata, which cannot fully capture document meaning in natural language queries [1], [2].

Recent advances in Artificial Intelligence (AI) [3], especially Natural Language Processing (NLP) [4], representation learning, and Large Language Models (LLMs) [5], provide a practical path toward semantic document understanding and automated knowledge access [6], [7]. In particular, Retrieval-Augmented Generation (RAG) [8] combines neural retrieval with generative reasoning, enabling systems to answer questions grounded in enterprise documents rather than relying only on parametric model memory. This architecture is highly suitable for internal knowledge systems where precision, provenance, and update ability are critical.

The proposed system integrates Optical Character Recognition (OCR) [9] enabled ingestion, semantic indexing with vector embeddings, document clustering,

role-based operations, and a chatbot interface for Vietnamese and mixed-language enterprise data. The solution is implemented as a full-stack platform (React + FastAPI + PostgreSQL/pgvector) [10] and evaluated through retrieval-quality experiments comparing baseline semantic search and clustering-enhanced search. The key contributions are:

- A practical end-to-end architecture for enterprise document digitization, indexing, retrieval, and AI-assisted question answering.
- A dynamic relevance-filtering mechanism based on K-Means clustering over semantic distances to reduce noisy retrieval outputs.

2. Related Work and Background

2.1. Document Intelligence System Overview

A document intelligence system is a software ecosystem that transforms unstructured and semi-structured documents into searchable, actionable organizational knowledge. In enterprise settings, this system usually spans the full lifecycle of document handling: ingestion, extraction, representation, retrieval, and decision support. Traditional document management platforms focus mainly on storage and keyword-based indexing, which are often insufficient for intent-level information access across heterogeneous files [2].

Modern document intelligence architectures add AI-driven capabilities to bridge this gap. First, OCR modules convert scanned pages and image-based PDFs into machine-readable text. Second, NLP pipelines segment and normalize content to create stable units for indexing. Third, embedding models map text chunks into dense vector spaces, allowing semantically similar documents to be located even when query and source wording differ [11]. Finally, RAG-based components connect retrieval outputs with LLM generation to produce context-grounded answers for enterprise users, with better quality and responsibility [12].

2.2. Existing Document Intelligence Techniques

Current document intelligence systems combine multiple techniques across the processing pipeline. At the acquisition stage, OCR remains essential for extracting text from low-quality scans, forms, and image-heavy documents. Although OCR performance has improved, noisy backgrounds, complex layouts, and multilingual content can still reduce extraction quality and introduce errors into downstream retrieval and QA stages.

At the semantic stage, dense representation learning has become the standard for indexing and similarity search. Transformer-based embeddings capture contextual meaning more effectively than sparse lexical features, supporting robust retrieval for paraphrased and semantically related queries [4]. In practice, these embeddings are typically searched through cosine similarity in vector databases or vector-enabled relational systems, often outperforming keyword-based methods in enterprise search.

At the reasoning stage, RAG is widely used to improve factual grounding in question answering. By incorporating retrieved evidence into the prompt, RAG reduces reliance on parametric memory and helps limit hallucinations [13], [14]. Still, top-k retrieval alone does not always provide sufficiently precise context. Meanwhile, implementation trends increasingly favor API-driven asynchronous backends and web-based interfaces, with modular frameworks and automated deployment supporting continuous AI system updates [15], [16]. Despite this progress, many systems still face challenges in balancing retrieval recall with precise context selection in enterprise corpora.

2.3. Motivation for Proposed Approach

Enterprise document systems still face three common problems: broad results from weak retrieval, noisy top-k chunks that reduce LLM answer quality, and the need for solutions that remain practical within standard web-stack environments [15].

To address these issues, this paper proposes an integrated architecture combining OCR-based ingestion, dense retrieval, RAG-based generation, and a clustering-based relevance filter before prompt construction. The filter improves context quality while preserving practical recall, which is especially important in enterprise QA settings where users prefer concise, evidence-grounded answers.

The approach also emphasizes deployment practicality. A unified stack using React, FastAPI, and PostgreSQL/pgvector supports both product functionality and research reproducibility [16]. Thus, the method serves as both a model-level improvement and a system-level strategy for turning document repositories into decision-ready knowledge services at enterprise scale [1].

3. Methodology

This section presents the methodological framework of the proposed enterprise document intelligence system. This also formalizes the retrieval and filtering mechanisms, including the K-Means-based relevance refinement stage, to clarify how the system improves answer quality and reduces noisy context before LLM generation.

3.1. Methodological Objectives and Design Principles

The methodology aims to build a document intelligence pipeline that is both scientifically grounded and practically deployable in enterprise environments. Rather than treating retrieval, semantic understanding, and question answering as separate tasks, the proposed approach unifies them into a retrieval-grounded workflow that improves downstream answer quality. Its main objective is to maximize relevance

and factuality under realistic enterprise constraints, including mixed document formats, noisy scans, and multilingual content [15].

The method is guided by three design principles: semantic-first retrieval to capture intent and conceptual similarity [11], grounded generation through RAG to condition outputs on retrieved internal evidence [13], and adaptive noise reduction through clustering before prompt construction. Together, these principles address common enterprise QA failure modes, particularly irrelevant evidence injection and hallucination amplification.

3.2. End-to-End Processing Pipeline

The full methodological flow consists of five sequential stages:

- **Document Ingestion and Normalization:** Uploaded files are validated and classified by type. Native text files are parsed directly, while scan-based PDFs and images are routed through OCR engines.

- **Semantic Chunking and Embedding:** Extracted text is segmented into semantically coherent chunks, then converted into dense vector embeddings via transformer-based encoders.

- **Vector Persistence and Retrieval:** Embeddings and metadata are stored in PostgreSQL/pgvector for similarity-based nearest-neighbor search.

- **Relevance Filtering via Clustering:** Retrieved candidates are refined through K-Means clustering on distance scores to separate likely relevant from noisy results.

- **RAG-Based Answer Synthesis:** Filtered context is injected into the LLM prompt, and the model generates evidence-grounded responses.

This staged design ensures traceability from raw document to final response while preserving modularity for future model substitution and scaling.

3.3. Mathematical Formulation of Retrieval and Generation

Let the enterprise corpus be $D = \{d_1, \dots, d_N\}$, where each document is decomposed into chunks $c_{i,j}$. For a user query q , embedding function $f(\cdot)$ maps text into dense vectors:

$$e_q = f(q), e_{i,j} = f(c_{i,j})$$

Similarity between query and chunk is computed by cosine similarity:

$$s(q, c_{i,j}) = \frac{e_q^T e_{i,j}}{\|e_q\|_2 \|e_{i,j}\|_2} \quad (1)$$

with T be the transpose operator.

A candidate set $R_k(q)$ is formed by selecting top- k chunks with highest s . The grounded generation objective is:

$$\hat{y} = \arg \max_y P_\theta(y | q, R_k(q)) \quad (2)$$

with token-level decomposition:

$$P_\theta(y | q, R_k) = \prod_{t=1}^T P_\theta(y_t | y_{<t}, q, R_k) \quad (3)$$

This formulation explicitly links retrieval quality and answer reliability, motivating the additional filtering stage before generation.

3.4. Dynamic Relevance Filtering with K-Means

Top- k semantic retrieval often includes marginally related chunks, especially when document collections are broad or query intent is underspecified. To address this issue, the method introduces a lightweight post-retrieval filtering step based on K-Means clustering over cosine-derived distance scores. In this study, $k = 2$ is selected because the filtering objective is binary: separating likely relevant chunks from weakly related or noisy chunks. Compared with using a fixed similarity threshold, this strategy adapts to the score distribution of each query and avoids manually tuning a single global cutoff for all retrieval cases. With $k = 2$, the optimization objective is:

$$J = \sum_{i=1}^k \sum_{x_n \in C_i} \|x_n - \mu_i\|_2^2 \quad (4)$$

where C_i and μ_i denote cluster set and centroid. The cluster with lower centroid is interpreted as high-relevance. If centroid separation is below a safety threshold δ , both clusters are retained to avoid over-pruning in ambiguous queries. This simple yet effective mechanism improves context purity without requiring expensive neural rerankers, aligning well with practical deployment constraints.

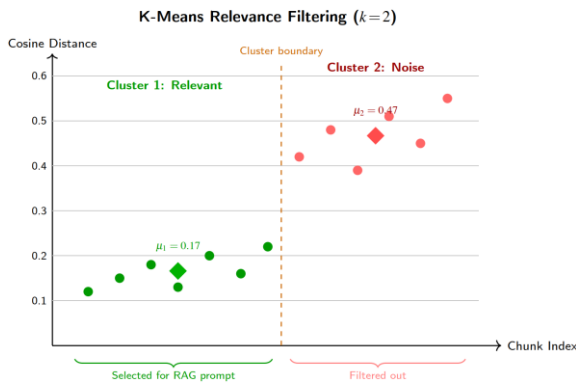


Figure 1. K-Means Relevance Filtering ($k=2$)

Figure 1 illustrates the filtering mechanism on a representative query. The horizontal axis indexes the top- k retrieved chunks, and the vertical axis shows their cosine distance to the query embedding. K-Means clustering ($k=2$) partitions the candidates into two groups: Cluster 1 (green, low centroid $\mu_1 = 0.17$) contains semantically close chunks selected for RAG prompt construction, while Cluster 2 (red, high centroid $\mu_2 = 0.47$) contains marginally related chunks that are filtered out. The dashed orange line marks the cluster boundary. This visual separation confirms that the method effectively isolates relevant context from retrieval noise without requiring expensive neural rerankers.

3.5. Algorithms Used in the Proposed Method

The proposed method is implemented through three connected algorithms corresponding to document ingestion, query-time retrieval, and relevance refinement. Together, these algorithms form an end-to-end pipeline that converts heterogeneous enterprise documents into indexed semantic representations and generates evidence-grounded responses for user queries.

Algorithm 1: Document Ingestion and Index Construction

Input: raw documents (PDF/image/text).

Output: indexed chunk embeddings with metadata.

- 1: Detect file type and validate upload policy.
- 2: Apply OCR for scan-based content; direct parsing otherwise.
- 3: Clean and normalize extracted text.
- 4: Segment text into semantically coherent chunks.
- 5: Compute dense embeddings for each chunk.
- 6: Store vectors + metadata in PostgreSQL/pgvector.

Algorithm 1 defines the offline indexing stage of the system. Uploaded documents are first validated and classified by format. Text is extracted either through direct parsing or OCR, depending on document type. The extracted content is then normalized, segmented into semantic chunks, embedded in vector space, and stored together with metadata.

Algorithm 2: Context-Aware Retrieval and RAG Answering

Input: query q .

Output: answer with retrieved evidence.

- 1: Embed query into vector space.
- 2: Retrieve top- k candidate chunks by cosine similarity.
- 3: Apply relevance filtering (Algorithm 3).
- 4: Construct grounded prompt from filtered context.
- 5: Generate response using LLM conditioned on context.
- 6: Return answer and supporting evidence snippets.

Algorithm 2 describes the query-time inference process. The user query is embedded into the same vector space as the indexed chunks, and the system retrieves the top- k semantically similar candidates. These candidates are then refined by the filtering stage before being used to construct the final prompt.

Algorithm 3: K-Means Relevance Filter

Input: candidate list with distance scores.

Output: refined context set.

- 1: Collect distance vector $X = \{x_1, \dots, x_m\}$.
- 2: Run K-Means with $k = 2$.
- 3: Identify low-centroid cluster as relevant set.
- 4: If $|\mu_1 - \mu_2| < \delta$, merge clusters.
- 5: Return filtered chunks for final generation.

Algorithm 3 is the main refinement stage of the proposed method. Because top- k retrieval may include marginally relevant chunks, K-Means clustering is applied to candidate distance scores rather than to the full embedding vectors. This design keeps the filtering step computationally lightweight and directly aligned with retrieval relevance. The cluster with the lower centroid is selected as the relevant set because it contains chunks closer to the query in semantic distance. When the centroid gap is smaller than the predefined threshold, both clusters are retained to avoid over-filtering in ambiguous queries. This fallback mechanism is important because some user questions may require evidence distributed across several moderately relevant chunks rather than a single compact relevance group.

Overall, the three algorithms implement the proposed document intelligence pipeline. Algorithm 1 constructs the semantic index, Algorithm 2 performs retrieval-augmented answering, and Algorithm 3 enhances context quality before generation. This design supports the methodological objective of improving retrieval precision and answer reliability in enterprise document environments.

3.6. System Interaction Flow

For paper illustration, the methodological flow can be visualized as two connected streams:

Ingestion stream: Upload -> OCR/Text Extraction -> Cleaning -> Chunking -> Embedding -> pgvector Storage

Query stream: User Query -> Query Embedding -> Vector Retrieval -> K-Means Filter -> Prompt Builder -> LLM -> Evidence-Grounded Answer

Figure 2 illustrates the complete methodology as two connected streams. The ingestion stream converts raw documents into indexed semantic representations through OCR, cleaning, chunking, embedding, and pgvector storage. The query stream transforms user queries into grounded answers through query embedding, vector retrieval, K-Means filtering, prompt construction, and LLM generation. Both streams are linked by the shared pgvector index, ensuring retrieval operates within the same semantic space built during ingestion.

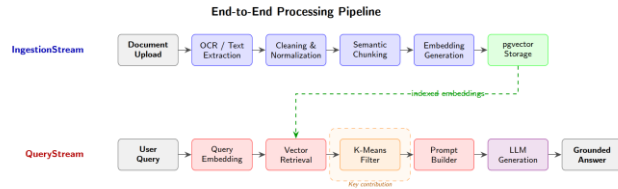


Figure 2. End-to-End Processing Pipeline

This diagram emphasizes that answer generation is never isolated; it is explicitly constrained by retrieved enterprise evidence, with clustering as a quality-control gate before final synthesis.

3.7. Evaluation Protocol and Methodological Validity

The methodology is evaluated by comparing two conditions under identical corpus and query sets:

- (i) baseline semantic retrieval,
- (ii) semantic retrieval plus K-Means filtering.

The evaluation corpus consists of representative enterprise-style documents collected from internal administrative, technical, procedural, and policy-related materials. The documents include both Vietnamese and mixed Vietnamese-English content, reflecting the practical language characteristics of organizational document repositories. After OCR/text extraction and preprocessing, the documents are segmented into semantic chunks and indexed in PostgreSQL/pgvector. The query set is manually designed to cover four common information needs: policy lookup, technical document retrieval, procedural guidance, and cross-document fact extraction. Relevance labels are assigned by checking whether each retrieved chunk contains sufficient evidence to answer the corresponding query. A chunk is considered relevant when

it directly supports the expected answer or provides necessary contextual evidence for answer generation.

Evaluation focuses on ranking quality and practical relevance concentration using MRR and Precision-like metrics:

$$\text{MRR} = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{\text{rank}_q}, \text{Precision@K} = \sum_{i=1}^K 1\{r_i \text{ relevant}\} \quad (5)$$

This protocol isolates the contribution of post-retrieval filtering and provides a reproducible basis for discussion in Section 5. The method is considered valid if it preserves strong ranking behavior while reducing irrelevant context returned to users and to the LLM prompt.

The proposed system operates on Vietnamese and mixed-language enterprise documents, which introduce challenges such as tonal diacritics, compound word segmentation, and more limited pre-trained model support than English. To address these issues, the system uses multilingual embedding models with Vietnamese support and applies Unicode normalization during text cleaning. In future works, we will explore Vietnamese-specific improvements, including syllable-aware chunking and domain-adapted embedding fine-tuning.

4. Implementation

This section outlines the practical implementation of the proposed architecture in a full-stack enterprise environment. It explains how frontend, backend, database, and AI components are integrated for document management, semantic retrieval, and RAG-based question answering. It also highlights deployment-oriented design choices related to modularity, access control, scalability, and maintainability.

4.1. System Architecture and Deployment Model

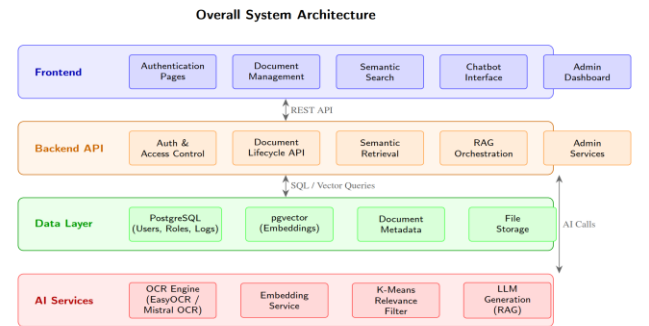


Figure 3. Overall architecture of the proposed enterprise document intelligence system

The architecture comprises four layers: (i) the Frontend layer providing role-based interfaces for document management, semantic search, and chatbot interaction; (ii) the Backend API layer implementing authentication, document lifecycle, retrieval, and RAG orchestration services; (iii) the Data Layer combining PostgreSQL for relational data and pgvector for dense embedding storage; and (iv) the AI Services layer integrating OCR engines, embedding generation, K-Means relevance filtering, and LLM-based response synthesis.

The proposed platform in Figure 3 is implemented as a modular client-server system to ensure scalability,

maintainability, and integration flexibility in enterprise environments. The architecture separates presentation, business logic, data persistence, and AI services, with communication performed through RESTful APIs [16]. This separation enables independent evolution of each component without disrupting the entire pipeline, which is important when updating embedding models, OCR engines, or LLM providers.

At deployment level, the system is organized into five core layers: (i) frontend interface, (ii) backend API gateway and service modules, (iii) relational and vector-enabled data layer, (iv) AI processing services, and (v) operational automation pipeline.

4.2. Frontend Implementation

The React frontend provides a responsive single-page interface for document workflows across customer, staff, and administrator roles. It supports authentication, document management, semantic search, chatbot interaction, and administrative monitoring, while its component-based design improves maintainability and extensibility.

The frontend sends asynchronous requests to backend services for retrieval and AI functions, avoiding heavy browser-side processing and keeping model orchestration centralized. It also presents generated answers with supporting context snippets to improve transparency and trust in enterprise workflows.

4.3. Backend Services and API Orchestration

Backend services are implemented with FastAPI for asynchronous performance, clear schema definition, and rapid API development. The service layer is organized into modules for authentication, document management, semantic retrieval, RAG orchestration, payment-ready operations, and administrative control, each exposed through versioned API routes.

Document processing is orchestrated by backend workers. On upload, the system validates files, extracts content, applies normalization and chunking, generates embeddings, and stores both metadata and vectors. At query time, it performs query embedding, vector retrieval, K-Means filtering, prompt construction, and grounded generation, ensuring that responses remain traceable to internal document evidence rather than unconstrained model outputs [13], [14].

4.4. Data Layer: PostgreSQL and pgvector Integration

The system uses PostgreSQL as the primary persistence engine for users, roles, access records, document metadata, and operational logs. To support semantic retrieval without introducing a separate vector engine, pgvector is integrated into PostgreSQL for dense embedding storage and nearest-neighbor search [17], [18]. This hybrid design simplifies deployment and reduces operational overhead for medium-scale enterprise scenarios.

Each indexed chunk is associated with structured metadata, including source document, processing timestamps, embedding model identifier, and confidence-related attributes when OCR/classification is applied. This design supports reproducibility and auditability during evaluation and production monitoring. In query execution,

cosine similarity ranking is computed directly at storage level, enabling fast retrieval and tight coupling with business constraints such as role-based document visibility.

4.5. AI Service Integration

The AI layer combines three functions: OCR, embedding generation, and LLM-based response synthesis. OCR services process scanned or image-based files to recover machine-readable content [19], [20]. Embedding services convert normalized chunks and incoming queries into dense vectors for semantic similarity operations [21]. The generation service uses a RAG workflow in which only filtered retrieval context is injected into prompts before answer generation [13], [22].

To improve practical response precision, the implementation includes a post-retrieval clustering stage (K-Means with $k = 2$) that separates highly relevant from weakly related candidate chunks prior to prompt composition. This module is implemented as a lightweight backend service and can be toggled for A/B comparison against baseline semantic retrieval. The result is a cleaner context window for generation, especially when top- k retrieval returns noisy tail matches.

4.6. Security and Access Control

The platform uses JWT-based authentication with role-aware authorization checks at API level. Customers access search and chatbot functions on permitted document scopes; staff receive extended upload and document handling privileges; administrators operate dashboard, user governance, department configuration, and payment management modules. This layered model aligns AI functionality with enterprise governance requirements and reduces risk of unauthorized knowledge exposure.

In addition, request validation, token expiry policies, and action logging are implemented to support operational traceability. These controls are particularly important for AI-assisted systems, where data provenance and controlled access must be preserved across both retrieval and generation stages.

5. Results and Discussion

This section presents the experimental and practical evaluation of the proposed enterprise document intelligence system. It focuses on validating the system from multiple perspectives, including retrieval effectiveness, functional operation, interface support, and real-world usability. Overall, this section aims to demonstrate that the proposed platform is not only technically implementable but also practically applicable to enterprise-oriented document management and AI-assisted information access.

5.1. Experimental Setup and Evaluation Scenario

The test set includes representative enterprise-style queries covering policy lookup, technical document retrieval, procedural guidance, and cross-document fact extraction. For each query, the system retrieves the same top- k candidates from vector storage under two settings: baseline semantic retrieval and semantic retrieval with K-Means filtering. In the baseline setting, all retrieved

candidates are forwarded directly to the RAG module. In the proposed setting, the K-Means filter removes weakly related candidates before prompt construction. The evaluation uses manually assigned relevance labels to compute MRR and Precision@k, while evidence grounding is assessed by checking whether the generated answer is supported by the retrieved context. This setup ensures that both methods are compared under identical corpus, query, embedding, and LLM conditions.

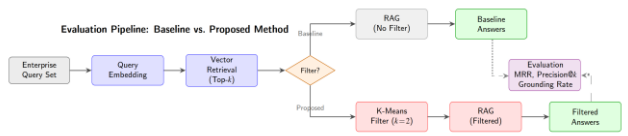


Figure 4. Evaluation pipeline comparing baseline semantic retrieval with the proposed K-Means-enhanced method

Both paths share identical query embedding and vector retrieval stages. The baseline forwards all top-k candidates directly to RAG, while the proposed method applies K-Means clustering (k=2) to filter noisy chunks before prompt construction. Both produce answers that are jointly evaluated using MRR, Precision@k, and evidence grounding metrics.

To ensure reproducibility, both settings use identical embedding models, identical LLM configuration [23], and identical prompt templates. Therefore, measured differences can be attributed primarily to the relevance filtering stage.

5.2. System Testing with Playwright

Functional verification was conducted via browser automation using Playwright, which simulated user interactions on the deployed platform to validate correct end-to-end execution from frontend to backend. Instead of reporting aggregate metrics, the evaluation reflects the exact test scope documented in the project report.

The automated workflow covered login page access, credential submission, authentication validation, message sending, and response retrieval. These steps capture the platform’s core interaction pipeline, particularly user authentication and AI-assisted communication, demonstrating that the system operates reliably beyond the interface level in a complete execution scenario.

Table 1. Comparison of automated test scenarios and verification objectives

Test scenario	User action	Verification objective	Observed scope
Login page access	Open the login interface	Authentication page is rendered correctly	Playwright test workflow
Credential submission	Enter credentials	Input and auth. request performed	Playwright test workflow
Auth. validation	Submit login request	The user can enter the system successfully	Playwright test workflow
Chat interaction	Send a message to the chatbot	The chatbot input pipeline can be triggered	Playwright test workflow
Response verification	Inspect the returned reply from the system	End-to-end interaction between user input and system	Playwright test workflow

Table 1 summarizes the automated test scenarios used to verify the core functional workflow of the proposed system. The results show that the testing process covers both authentication-related operations and chatbot-based interaction. Overall, the table confirms that the main user actions, from login to response retrieval, were successfully validated through the Playwright testing workflow.

5.3. Representative Interface Results

This subsection presents representative interface-level results of the implemented platform. These interface results are important because they show that the proposed system was not limited to a conceptual architecture or isolated backend services; instead, it was realized as a complete application with multiple operational views.

Table 2 presents the distribution of key system functions across different user roles. This comparison helps clarify how the proposed platform applies role-based access control to separate general usage, operational tasks, and administrative responsibilities.

Table 2. Comparison of interface support across user roles

Feature group	Customer	Staff	Admin
Authentication	Yes	Yes	Yes
Document upload	Yes	Yes	Yes
Bulk upload	No	Yes	Yes
Document security	No	Yes	Yes
Semantic search	Yes	Yes	Yes
Chatbot interaction	Yes	Yes	Yes
User management	No	No	Yes
Department management	No	No	Yes
Payment management	No	No	Yes

Table 2 shows that the proposed system applies a clear role-based access structure across its main interface functions. Core features are available to all users, whereas operational and administrative functions are assigned to staff and admin roles according to their responsibilities. Overall, the table indicates that the platform is designed to support both usability and access control in an enterprise setting.

5.4. Discussion of Strengths, Limitations, and Practical Implications

The proposed system integrates document ingestion, OCR/text extraction, semantic indexing, and retrieval-augmented answering within a modular enterprise architecture. PostgreSQL/pgvector stores chunk embeddings with source metadata, linking semantic retrieval to document management. The K-Means filter operates on candidate distance scores rather than full embeddings, providing a post-retrieval context selection step without an additional learned reranking model.

The filter groups candidates according to each query’s score distribution, while using a predefined centroid-separation threshold to limit over-filtering. As described in Algorithm 3, both clusters are retained when their centroids are insufficiently separated. However, semantic proximity does not guarantee sufficient supporting evidence, and the

two-cluster assumption may exclude useful context for questions requiring information across multiple chunks. OCR errors and chunk boundaries remain additional limitations, as filtering cannot recover information lost during extraction or segmentation.

Our architecture targets internal document workflows where users retrieve information within permitted scopes and inspect supporting source snippets. Source metadata and role-aware access controls support traceability and controlled access, but do not ensure answer correctness. The modular filtering stage also provides a basis for investigating adaptive selection strategies and domain-specific reranking without redesigning the ingestion and storage pipeline.

6. Conclusion

This paper presents an AI-enabled digital transformation system for intelligent document management, integrating OCR, dense semantic retrieval, pgvector-based indexing, and RAG-driven question answering in a full-stack deployment. Beyond baseline semantic search, the study introduces a clustering-based relevance filter that improves result focus and practical precision while maintaining strong ranking behavior. The implementation demonstrates feasibility for real organizational workflows, including multi-role operations, secure API-based access, and production-oriented deployment practices. Experimental observations confirm that combining semantic search with dynamic filtering produces cleaner retrieval context and more reliable downstream chatbot outputs. Future work will prioritize large-scale performance benchmarking, adaptive clustering/reranking strategies, advanced security controls, and domain-specialized AI agents for legal, education, and technical corpora. These directions can further strengthen enterprise knowledge access and accelerate comprehensive digital transformation.

Acknowledgments: This research is funded by Funds for Science and Technology Development of the University of Danang under project number B2025-DN02-25.

REFERENCES

- [1] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer, 2006.
- [2] A. K. Jain, "Data clustering: 50 years beyond K-means", *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010.
- [3] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th edition. Harlow, UK: Pearson, 2021.
- [4] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*, 3rd ed., Stanford University, Stanford, CA, USA, 2026. [Online]. Available: <https://web.stanford.edu/~jurafsky/slp3/> [Accessed March 12, 2026].
- [5] W. X. Zhao *et al.*, "A survey of large language models", arXiv preprint arXiv:2303.18223, 2023.
- [6] scikit-learn Developers, "Clustering: K-Means", *scikit-learn documentation*, 2025. [Online]. Available: <https://scikit-learn.org/stable/modules/clustering.html#k-means> [Accessed March 12, 2026].
- [7] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space", arXiv preprint arXiv:1301.3781, 2013.
- [8] W. Fan *et al.*, "A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models", in *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, Barcelona, Spain, 2024, pp. 6491–6501. <https://doi.org/10.1145/3637528.3671470>
- [9] S. V. Rice, G. Nagy and T. A. Nartker, *Optical Character Recognition: An Illustrated Guide to the Frontier*. Boston, MA: Springer, 1999.
- [10] pgvector Developers, "pgvector: Open-source vector similarity search for Postgres", *GitHub repository*, 2026. [Online]. Available: <https://github.com/pgvector/pgvector> [Accessed March 12, 2026].
- [11] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks", arXiv preprint arXiv:1908.10084, 2019.
- [12] Google DeepMind, "Gemma: Open models for responsible AI", 2024.
- [13] OpenAI, "Text embeddings guide", *OpenAI Platform Documentation*, 2025. [Online]. Available: <https://platform.openai.com/docs/guides/embeddings> [Accessed March 12, 2026].
- [14] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks", in *Advances in Neural Information Processing Systems*, 2020.
- [15] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, Doctoral dissertation, University of California, Irvine, Irvine, CA, USA, 2000.
- [16] M. Richards, *Software Architecture Patterns*. Sebastopol, CA: O'Reilly Media, 2015.
- [17] S. Ramírez, "FastAPI framework, high performance APIs with Python", *FastAPI Docs*, 2025. [Online]. Available: <https://fastapi.tiangolo.com> [Accessed March 12, 2026].
- [18] PostgreSQL Global Development Group, "PostgreSQL 16 Documentation", *PostgreSQL.org*, 2025. [Online]. Available: <https://www.postgresql.org/docs/16> [Accessed March 12, 2026].
- [19] Jaided AI, "EasyOCR: Ready-to-use OCR with 80+ languages", *GitHub*, 2025. [Online]. Available: <https://github.com/JaidedAI/EasyOCR> [Accessed March 12, 2026].
- [20] Mistral AI, "OCR", *Mistral Docs*, March 6, 2025. [Online]. Available: <https://docs.mistral.ai/models/ocr-25-03> [Accessed March 12, 2026].
- [21] A. Singhal, "Modern information retrieval: A brief overview", *IEEE Data Engineering Bulletin*, vol. 24, no. 4, pp. 35–43, 2001.
- [22] O. Khattab and M. Zaharia, "ColBERT: Efficient and effective passage search via contextualized late interaction", in *Proc. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2020, pp. 39–48.
- [23] N. N. H. Van, P. H. Do, V. N. Hoang, T. T. K. Nguyen and M. T. Pham, "AI-Powered University Admission Counseling: A Use Case of Large Language Models in Student Guidance," in *IEEE Transactions on Learning Technologies*, vol. 18, pp. 856–868, 2025, doi: 10.1109/TLT.2025.3604096.