Available online at website: https://jurnal.iaii.or.id/index.php/RESTI

JURNAL RESTI

(Rekayasa Sistem dan Teknologi Informasi)

Vol. x No. x (2026) xx - xx e-ISSN: 2580-0760

Data Extraction and Agent Orchestration using LangGraph on a Chatbot Retrieval-Augmented Generation (RAG) System for mySRE

Robiatul Adawiah[1], Rio Nurtantyana[2], Aria Bisri[3], Iman Wahyudi[4]

[1]Informatics, Faculty of Science & Technology, UIN SMH Banten, Banten, Indonesia

[2]Research Group HCI-V, Data and Information Science, Center,National Research and Innovation Agency, Bandung, Indonesia

[3]Research Group HCI-V, Data and Information Science, Center,National Research and Innovation Agency, Bandung, Indonesia

[4]Informatics, Faculty of Science & Technology, UIN SMH Banten, Banten, Indonesia

*robiatuladawiahayu@gmail.com,

Abstract

Academics seeking to locate and retrieve data face a major difficulty in keeping up with the always expanding volume of scientific research. Large Language Models (LLMs) provide a user-friendly approach to engage with knowledge, but their constraints include limited datasets and the possibility of producing inaccurate results. To solve this, our research introduces MySRE, an intelligent RAG (Retrieval-Augmented Generation) system. It uses LangGraph to manage a complex mix of graph and vector databases. MySRE uses the PRAL (Perceive, Reason, Act, Learn) framework to process academic papers by integrating PostgreSQL (with pgvector) and Neo4j, so organizing them into essential knowledge categories such Background, Methodology, Research Gap, Objectives, and Future Work. MySRE increases cognitive accuracy to 75.00% and reaches an F1-Score of 76.92%, as indicated by our tests. This greatly outperforms a normal ChatGPT model that directly processed PDFs, which scored 66.67% for both accuracy and F1-Score. Though the LangGraph orchestrator's decision-making process results in a small rise in response latency (16.44 s vs. 11.12 s), MySRE lowers operational API expenses by about eight times (0.00034867 vs. 0.00281068 per query) by using context compression. These findings support the accuracy and scalability of agent-based hybrid RAG systems in intelligent research settings.

Keywords: retrieval augmented generation1; langgraph2; pgvector3; neo4j4; agentic ai5

How to Cite: [Caption completed by the editor]
*Permalink/*DOI: [Caption completed by the editor]

Received: [Caption completed by the editor]
Accepted: [Caption completed by the editor]
Available Online: [Caption completed by the editor]

1. Introduction

The large number of scientific articles coming out nowadays suggests that there is a staggering quantity of research available. For scientists, this is amazing since it provides them with fresh information from across the board. But it also makes it difficult to really locate and apply the data most relevant to their jobs. If researchers are doing it all by hand, it is typical for them to spend hours sifting through numerous papers just to find what they need for their studies, therefore slowing down the literature review process.

The development of Large Language Models (LLMs) offers a new approach to assisting the information retrieval process through AI-powered chatbots. In a study conducted by Hamideh Ghanadian, Amin Kamali, and Mohammad Hossein Tekieh (2026), the authors state that "In the scientific domain, chatbots equipped with retrieval-augmented generation (RAG) capabilities can significantly enhance literature reviews by retrieving and summarizing relevant research papers[1]." These capabilities make chatbots potentially useful as research assistants that can help users obtain information through natural language interaction.

However, the standalone use of LLMs still has several limitations. The knowledge possessed by the model is static and depends on the training data used during the training process. In a study conducted by Sangita Pokhrel, Bina K.C., and Prashant Bikram Shah (2024), the authors state that "The fact that their knowledge base is fixed and set at the time of training, however, is their primary drawback[2]." As a result, the model does not always have the latest information and may generate answers that do not align with the facts (hallucinations).

To address these limitations, the Retrieval-Augmented Generation (RAG) approach was developed, which combines retrieval and generation processes within a single framework [3], [4], [5]. This architecture was originally formalized by Lewis et al., who combined a parametric generator with a non-parametric retrieval index to improve performance on knowledge-intensive tasks[5]. Recent studies have highlighted RAG as an effective approach for enhancing the factual accuracy and contextual relevance of Large Language Models by integrating external knowledge sources during inference [4], [6]. In a study conducted by Sangita Pokhrel, Bina K.C., and Prashant Bikram Shah (2024), the authors state that "The Retrieval-Augmented Generation (RAG) architecture was developed to overcome this limitation by integrating external knowledge retrieval systems with large language models (LLMs)[2]." Through this approach, the system can retrieve information from external sources, enabling it to generate more relevant and accurate answers while reducing the risk of hallucinations[4], [6], [7]. Gao et al. similarly categorize RAG systems into Naive, Advanced, and Modular paradigms based on the complexity of their retrieval and augmentation mechanisms[8]

The quality of the data used as the knowledge base determines in large measure how well RAG works. Usually documents make up the knowledge sources in a Smart Research Environment, or SRE. These files include mathematical equations, tables, graphics, and text, among other things. Before it can be utilized to locate what we are searching for, all of this data has to be retrieved. Should we extract this data incorrectly, we might overlook items. This implies the RAG system's search and response quality will suffer. If the data is not decent, the RAG execution will not be effective.

In addition, the majority of RAG systems still depend on linear workflows. Our system has to dynamically evaluate queries, select pertinent context, retrieve data, and produce responses. Thus we need a means to oversee all these stages somewhat. LangGraph uses a graph-based method for controlling workflows. This method lets agent processes be managed in a adaptable and dynamic manner. The graph-based method allows agents to operate more efficiently. It supports management of every phase of the process in a methodical way. LangGraph enables flexibility to meet evolving requirements. It enables efficient handling of agent work processes. This strategy could help RAG systems. The performance of RAG systems can be enhanced by using LangGraphs approach.

This study suggests creating a system based on these difficulties that retrieves data and controls agents using LangGraph on a Chatbot. This Chatbot integrates a Retrieval-Augmented Generation (RAG) System to facilitate a Smart Research Environment (SRE). The study concentrates on obtaining document data. LangGraph is also used to control agents for query analysis, information retrieval, and response generation. This will raise retrieval quality of the system. Inside the Smart Research Environment, it will produce pertinent answers to aid research activity.

2. Methods

The methods The architecture of the MySRE system is discussed in this chapter. There are a few components in the design of the MySRE system. It has a data ingestion pipeline and a way to map things to two databases. The design of MySRE also incorporates flow orchestration and a means for users to engage with it via chatbots. MySRE system design even handles memory to enable it to recall objects.

The designers of the MySRE system meant for it to manage the data it receives in an appropriate manner. They also aimed to improve the speed with which the MySRE system design reacted to consumers. The people who designed the MySRE system were worried about hallucination risks, which can happen in language models or LLMs. They looked for approaches within the architecture of the MySRE system to lower these hazards.MySRE

2.1 System architecture and PRAL cognitive cycle

The MySRE system works with something called the PRAL cycle. The PRAL cycle is, like a circle that the MySRE system follows. It does four things:

Perceive: the MySRE system takes in information

Reason: the MySRE system thinks about what it has learned

Act: the MySRE system does something based on what it knows

Learn: the MySRE system tries to get better at what it does

The MySRE system uses the PRAL cycle to control how the chatbot behaves. This helps the MySRE system work efficiently and not say things that're not true. The PRAL cycle is important for the MySRE system to work properly with the chatbot behavior.

Table 1. PRAL cycle

STAGE	Core Activity	Output
Perceive	Reads the user query, conversational history, and list of active document hashes	Initial system state (PRALState).

Reason	Classifies the user query using an LLM to determine the retrieval pathway.	Retrieval layer selection (1, 2, or 3) and intent classification.
Act	Executes semantic search, expands graph relations, compresses context, and generates the response.	Selected context chunks and Markdown-formatted response.
Learn	Logs the conversation details and system execution metrics.	Updated chat history and performance metrics.

This cyclical design follows the broader trend in agentic AI research, where large language models function as reasoning engines within a structured perceive-reason-act-learn loop rather than operating as single-pass, reactive systems[9]. This interleaving of reasoning traces and actions was first demonstrated by Yao et al., who showed that alternating thought and action steps improves an LLM's ability to plan and recover from errors[10]. This cycle ensures that all document retrieval actions and response generation steps are guided by the routing decision made during the *Reason* stage.

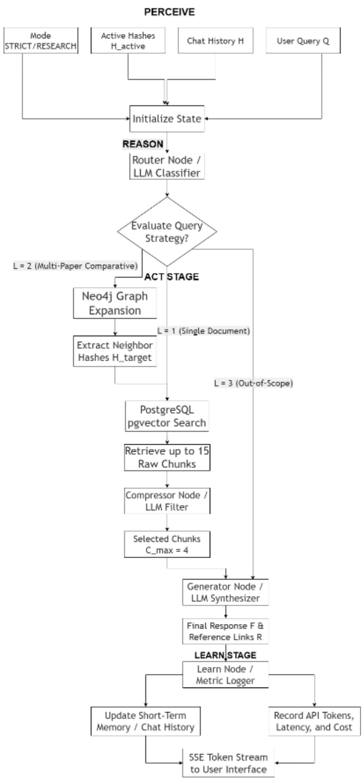

Fig. 1. Flow diagram of the agentic PRAL cognitive cycle in MySRE.

As illustrated in Fig. 1, the PRAL cognitive cycle governs the state transitions and execution paths dynamically. In the *Perceive* stage, the system initializes the state by aggregating user input query Q, conversational history H, active document hash list $H_{activate}$, and UX mode configuration. During the *Reason* stage, the RouterNode acts as a zero-temperature LLM classifier, analyzing the query to resolve the retrieval layer strategy L. In the *Act* stage, if the query is out-of-scope (L=3), the orchestrator routes the execution flow directly to the GeneratorNode via a short-circuit pathway, bypassing all database queries. For in-scope queries $L \in \{1,2\}$, the system runs the retrieval pipeline. If comparative analysis is required $L = 2$, the graph expansion in Neo4j identifies neighboring documents and generates a target hash list H_{target} prior to semantic search. PostgreSQL performs cosine similarity vector searches across H_{target} to extract up to 15 raw chunks. The Compressor Node then filters these chunks to a maximum of $C_{max} = 4$ highly relevant chunks before the GeneratorNode synthesizes the final response. Finally, in the *Learn* stage, the LearnNode logs operational metrics and updates the short-term conversation memory to prepare for the subsequent user query.

2.2 Data ingestion and extraction pipeline

The data pipeline handles scientific papers and turns them into numbers that computers can understand. This helps with a process called retrieval-augmented generation or RAG for short. The data pipeline takes in scientific literature. It then processes this literature. The goal is to turn it into numerical representations. These representations are suitable for retrieval-augmented generation, which's what RAG is all, about.

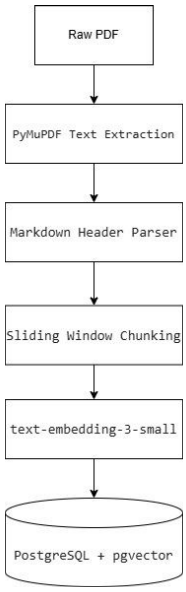

Fig.2 flowchart

1. Sliding Window Chunking

Text is extracted from raw PDF documents using PyMuPDF (fitz), preserving headings, paragraph breaks, and layout structures. The extracted text is then decomposed into smaller blocks using a sliding window algorithm implemented via MarkdownHeaderTextSplitter and RecursiveCharacterTextSplitter. Let the raw document text be represented as a sequence of characters $T = \{t_1, t_2, \dots, t_M\}$ with a total length of M characters. The chunking process generates a set of chunks $C = \{c_1, c_2, \dots, c_N\}$ with a fixed window size $K = 1500$ characters and an overlap size $0 = 150$ characters. The start index S_i and end index E_i for each chunk i are calculated as follows:

$$S_i = (i - 1 \times (K - 0) + 1$$

$$E_i \min(S_i + K - 1, M)$$

Where $i \in [1, N]$ and the total number of chunks N is defined as:

$$N = \left\lceil \frac{M - 0}{K - 0} \right\rceil \qquad (2)$$

The overlap $0 = 150$ characters is maintained to prevent semantic fragmentation at chunk boundaries, This design choice aligns with prior findings showing that chunk size and overlap configuration substantially influence the trade-off between contextual completeness and retrieval precision in RAG pipelines[11], ensuring the LLM receives complete sentences and contexts.

2. Vector Embedding Generation

Each chunk $c_{i \backslash inc}$ is mapped to a high-dimensional vector representation $V^{d}_{i \backslash inR}$ $(d = 1536)$ using the text-embedding-3-small model:

$$v_i \mathrm{Embed}(c_i)$$

This embedding model builds on prior work demonstrating that contrastive pre-training at scale yields high-quality vector representations suitable for semantic search[12]. Dense retrieval methods have been shown to outperform traditional lexical search for semantic similarity tasks[13]. This is consistent with prior work demonstrating that fine-tuned sentence embedding models enable efficient cosine-similarity comparison at scale[14]. The embedding vector v_i captures contextual semantic relationships, allowing the system to perform searches based on conceptual meaning rather than exact keyword matching.

3. Evolution and comparative evaluation of chunking strategies

Before selecting the final Header-Aware Sliding Window Chunking strategy, five different chunking methods were experimentally evaluated to determine the optimal configuration for the MySRE hybrid architecture. Table 2 summarizes the comparative features of these evaluated strategies.

Table 2. Comparative analysis of evaluated chunking strategies

Strategy	**Computational Latency**	**API Dependency**	**Context Coherence**	**Neo4j Integration**	**Rate-Limit Resilience**
Fixed-Size Linear	Extremely Fast	None (Offline)	Low (Breaks sentences)	Difficult	High
Semantic similarity	Very slow	High (Calculates sentence distances)	High (Topic-based)	Moderate	Low
Gemini 2.5 Structured Output	moderate	High (External LLM Schema API)	High (Pillar-based)	Moderate	Low (API Bottleneck)
Hierarchical Parent-Child	slow	Moderate (Dual-pass vector mappings)	High (Hierarchical)	Complex	Moderate
Header-Aware Sliding Window	Fast	None (Offline parsing)	High (Maintains headers)	Seamless (Direct Mapping)	High (100% Offline)

a. **Fixed-Size Linear Chunking:** Documents are sliced blindly based on character counts. This approach is computationally efficient but often splits key scientific arguments mid-sentence, introducing semantic noise and losing organizational context (e.g., mixing sections).
b. **Semantic Similarity Chunking:** Splits are calculated dynamically based on cosine distance thresholds between adjacent sentence embeddings. Although it keeps coherent topics together, the computational cost of generating sentence-level embeddings offline is extremely high, making it unscalable for large libraries.
c. **Structured Ingestion via Gemini 2.5 Flash:** PDF text is parsed by Gemini 2.5 Flash and mapped directly to the five scientific pillars using JSON Schema *Structured Output*. However, using a cloud API for bulk document preprocessing introduced a severe throttling bottleneck due to API rate limits on the free-tier service.
d. **Hierarchical Parent-Child Chunking:** Splits the text into large parent chunks (1500 characters) and smaller child chunks (200 characters) linked via unique identifiers. While effective for localized semantic searches, retrieving double-mapped relationships inside Neo4j and PostgreSQL introduced high database query complexity and processing latency.
e. **Selected Strategy (Header-Aware Sliding Window):** Combines MarkdownHeaderTextSplitter (which parses headers like #, ## to attach section contex as metadata)with RecursiveCharacterTextSplitter (window size K=1500 overlap O=150). This approach runs entirely offline, provides complete sentence coherence, and allows seamless node mapping in the Neo4j knowledge graph.

2.3. Dual Database Mapping

MySRE implements a hybrid storage architecture to optimize search operations. This approach isolates vector semantic representations from document relation mappings.

1. *PostgreSQL with pgvector*

The embedding vector v_i is stored in a PostgreSQL database enabled with the pgvector extension. The table index is optimized using a Hierarchical Navigable Small World (HNSW) index based on cosine distance. The HNSW index parameters are configured with M_{16} and $ef_{construction} = 64$ to balance query speed and retrieval accuracy. This indexing method is based on the hierarchical graph-based approximate nearest neighbor search algorithm originally introduced by Malkov and Yashunin, The rapid emergence of vector database management systems in recent years has been driven largely by LLM-based applications requiring scalable similarity search over unstructured data [15]. which organizes vectors into multiple layers of proximity graphs to accelerate similarity search without exhaustively scanning the entire dataset [16].

Table 3. DocumentChunk table schema in PostgreSQL

Column name	Data type	Description
id	SERIAL / UUID	Unique identifier for each document chunk.
composite_id	VARCHAR(64)	Deterministic MD5 hash representing tenant isolation.
file_hash	VARCHAR(32)	MD5 hash of the source PDF file.
title	TEXT	Source document title or file name.
content	TEXT	Raw text content of the chunk.
metadata_header	TEXT	Header path hierarchy associated with the chunk.
embedding	VECTOR(1536)	Semantic vector embedding of the chunk text.

To enforce row-level security and multi-tenant isolation, each chunk record is bound to a deterministic *Composite ID* generated using the MD5 hashing function:

$$Composit\,e_{ID} = MD\left(User_{ID}||Project_{ID}||File_{Hash}\right)$$

Where the symbol || denotes string concatenation. This mechanism prevents data leaks across different user projects by isolating documents at the query level.

2. *Neo4j Graph Database*

Knowledge graphs can complement LLMs by supplying explicit, structured factual knowledge that black-box language models cannot natively access or verify[17]. Metadata and relationship networks are mapped into a Neo4j graph database, formally represented as $G = (V, E)$ where:

a. V is a set of nodes representing scientific documents (*D*) or SRE knowledge pillars (*P*).

b. *E* is a set of directed edges representing relationships (e.g., CONNECTED_TO or HAS_PILLAR).

A Document node contains properties such as hash (**file identifier**), title (**document title**), project_id (**project ID**), and user_id (**user ID**). A Pillar node contains properties like name (*e.g., Objective, Methodology, Background, Gap, Futurework*).
The CONNECTED_TO relationship between two documents is automatically generated by the system's reasoning agent when a strong semantic connection is detected. This relationship stores a reasoning property explaining the linkage, along with a similarity score.

2.4 Agentic flow orchestration based on LangGraph

The chatbot workflow is managed using **LangGraph**. The execution flow, decision-making, context assembly, and response formatting are modeled as a directed graph (*StateGraph*). Graph-based retrieval approaches have also been shown to support multi-hop, cross-document reasoning by building an entity-level knowledge graph prior to retrieval [18].

1. *Formal representation of agent state (PRALState)*

The LangGraph workflow operates on a shared state updated progressively by each node. The state is represented as a tuple S*S*:

$$S = \langle Q, P, U, H_{activate}, M, L, I, H_{target}, D_{neighbor}, G_{context}, C_{raw}, C_{comp}, R, F, M_{sta}$$

Where the elements are defined as follows:

a. $Q \in \Sigma$: The initial user query.

b. $P \in \Sigma$: The project identifier.

c. $U \in \Sigma$: The user identifier.

d. $H_{activate} \subset \Sigma^{23}$: Himpunan of active document hashes.

e. $M \in \{STRICT, RESEARCH\}$: System execution mode.

f. $L \in \{1, 2, 3\}$: The selected retrieval layer.

g. $I \in \Sigma^*$: Classified search intent.

h. $H_{target} \subset \Sigma^{32}$: Target document hashes.

i. $D_{neighbor}$: Structures representing connected neighbor documents in Neo4j.

j. $G_{context} \in \Sigma^*$: Joined graph relation descriptions from Neo4j.

k. C_{raw}: Retrieved raw text chunks from PostgreSQL.

l. C_{comp}: Selected compressed chunks.

m. R: Reference links payload formatted as JSON.

n. $F \in \Sigma^*$: Final response in Markdown.

o. M_{stats}: Performance metrics (execution latency, token counts).

2. *Node system mechanics*

The orchestrator consists of four functional nodes:

a. Router Node (Reason stage)

This node evaluates the user query Q using gpt-4o-mini with a temperature of $\tau = 0.0$ to ensure deterministic routing. The underlying gpt-4o-mini model, like other modern LLMs, is built upon the Transformer architecture originally introduced by Vaswani et al.[19]. Contextual representations derived from pretrained transformer encoders such as BERT laid the groundwork for modern embedding-based retrieval [20].The query is classified into one of three retrieval layers:
Layer 1 (Single Active Document): Triggered when the query focuses on the contents of a single active document. Layer 2 (Multi-Paper Comparative): Triggered when the query requires comparative analysis or cross-document relationship discovery. Layer 3 (General/Out-of-Scope): Triggered when the query is unrelated to the active literature.

Formally, the node updates the state as follows:

$$RouterNode(s) \rightarrow s[L \leftarrow Layer\ Class, I \leftarrow intent\ Text]$$

b. Retrieval Node (Act stage)

If $L \in \{1,2\}$ the node executes hybrid retrieval:

Graph Expansion (Neo4j): If $L = 2$, the node queries Neo4j asynchronously to find neighbor documents connected to the active document hash list. Up to $N_{neighbor}$ connected documents are retrieved. Their relationship reasons are appended to $G_{context}$ and their hashes are added to H_{target} **Semantic Search (PostgreSQL):** Computes the cosine similarity between the query embedding q and chunk embeddings vjvj across target documents in H_{target} :

$$\text{Similarity}(q, v_j) = \frac{q \cdot v_j}{\|q\|_2 \|v_j\|_2} = \frac{\sum_{k=1}^{d} q_k v_{j,k}}{\sqrt{\sum_{k=1}^{d} q_k^2}\sqrt{\sum_{k=1}^{d} v_{j,k}^2}}$$

The node retrieves up to K_{vector} =15 raw chunks with the highest scores, saving them to C_{raw}.

c. *Compressor Node (Act stage)*

To optimize prompt size, raw chunks in C_{raw} are filtered by the Compressor Node. The node uses gpt-4o-mini to evaluate chunk relevance to query Q, retaining a maximum of $C_{max} = 4$chunks in C_{comp}. This compression step minimizes input token usage while preserving information quality.

d. *Generator Node (Act stage)*

This node generates the final response F using the selected chunks $C_{camp}$and the system's *Research Policy*. The output is cleaned of raw markdown tags, and citation links are appended dynamically to the response text.

2.5 Conditional routing logic

The conditional routing mechanism in LangGraph optimizes execution time and API cost. State transitions depend on layer classification variable *L* resolved by the Router Node.

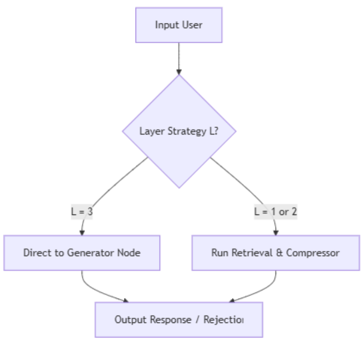

Figure.3 conditional Routing Flowchart

Short-circuit route (L = 3): If the Router Node classifies the query as out-of-scope (Layer 3), the workflow skips PostgreSQL and Neo4j queries, routing directly to the Generator Node. The generator processes the query using pre-configured system rules. This pathway reduces database search latencies to zero. **Full RAG route (L = 1 or L = 2):** The graph executes the full sequence (Router → Retrieval → Compressor → Generator) to gather factual evidence from target documents before formulating the response.

2.6 Chatbot interaction and memory management

This section outlines how the chatbot maintains communication flow and session context.

1. *Asynchronous token streaming via Server-Sent Events (SSE)*

To make things feel faster the system uses a way of sending tokens one after the other without waiting for each other. When the Generator Node is working it uses a method called astream from the LLM to send out text tokens a little at a time. The LLM sends these tokens to the user interface using something called Server-Sent Events, which's, like a messenger that helps show the user what is happening right away.

2. *Short-term memory management*

The conversation history is taken care of by the ChatHistoryManager. It keeps track of what the user asks and what the assistant says. The ChatHistoryManager also keeps links that're relevant and how long it takes to get a response. When someone asks a question the ChatHistoryManager adds the last few things that were said to the prompt. This helps the language model understand what is being talked about when it sees words like "this" or "that". The ChatHistoryManager only adds the three things that were said so the conversation does not get too long. This way the language model can understand what the user means by "this document" or "the previous method”, without getting confused.

3. *UX mode guardrails*

The interface enforces interaction rules based on the user's active document configuration:

a. *STRICT Mode (No Active Document):* If the system detects that the active hash list is empty (Hactive=∅), the workflow is blocked, and the chatbot displays an instruction prompting the user to select a document from the graph canvas.

b. *STRICT Mode (Active Document, Multi-Document/General Query):* If a user submits a comparative query (L=2) or general query (L=3) in STRICT mode, the chatbot prompts the user to switch to RESEARCH mode to enable database graph expansion.

c. *RESEARCH Mode (No Active Document):* The system permits general scientific discussions (L=3) but displays a prompt advising the user to activate documents to access graph visualization features.

3. Results and Discussions

Two systems are examined in this chapter: MySRE, which blends graph and vector approaches, and the Baseline model, which employs ChatGPT with Direct

PDF. The assessment is determined by the quality of the information's To assess data quality, we looked into the DeepEval framework. To find out how well the systems categorize items, we also utilized a confusion matrix. We also investigated how well the systems operate and how much it costs to operate them by means of APIs. Regarding MySRE's method and Baseline's utilization of ChatGPT, the MySRE system and the Baseline model are contrasted. On the basis of these parameters, the performance of the MySRE systems and the Baseline models are assessed. MySRE and Baseline are contrasted to ascertain which one is more effective.

3.1 Experimental setup and testing scenarios

The experiment evaluates system performance in understanding SRE scientific literature and detecting out-of-scope queries. The testing parameters were configured identically for both systems:

1. *Language model specification:* OpenAI's gpt-4o-mini API was utilized for both setups. The Router Node was configured with a low temperature ($\tau = 0.0$) to guarantee deterministic routing, while the Generator Node used a moderate temperature ($\tau = 0.3$) to allow linguistic flexibility under the system's research policy.
2. *Embedding model specification:* The text-embedding-3-small model was used to generate 1536-dimensional vector embeddings.
3. *Testing dataset (evidence pack):* The evaluation consists of 24 execution sessions based on 12 unique queries across 2 scientific PDF documents. The first document is a qualitative-descriptive case study on cash waqf collection and management strategies, and the second is a qualitative historical-political study of Islamic politics during the Old Order era.The queries are categorized into three groups:

   a. *Single-document queries (in-scope):* Questions where answers are directly present in a single active document (queries 1-5).

   b. *Comparative queries (in-scope):* Questions requiring the synthesis of information across different sections of the document (queries 6-8).

   c. *Out-of-scope queries (adversarial):* Trick questions on topics not discussed in the source documents (e.g., World War II history, current President of Indonesia, and Einstein's theory of relativity) to test the system's resistance to hallucination (queries 9-12).

The MySRE system was evaluated with a chunk size of $K = 1500$ characters and an overlap of $O = 150$ characters, retrieving the top 15 chunks via PostgreSQL, expanding up to 3 connected documents in Neo4j, and compressing context to a maximum of 4 chunks. The Baseline system processed the entire PDF file directly without database retrieval.

3.2 Classification accuracy analysis (Confusion matrix)

Cognitive accuracy is evaluated using a confusion matrix classified under the following definitions:

1. *True Positive (TP):* The system answers an in-scope question accurately (relevancy score ≥0.5, faithfulness score ≥0.5).
2. *True Negative (TN):* The system correctly identifies and rejects an out-of-scope question.
3. *False Positive (FP):* The system generates a response to an out-of-scope question using general knowledge or hallucinates factual information.
4. *False Negative (FN):* The system fails to answer an in-scope question, or generates a response with relevancy below 0.5.

Table 3 summarizes the classification metrics compiled from the test logs:

Table 4. Confusion matrix classification summary

Model (Method)	average_char_input	average_char_output	average_latency	average_cost
Baseline (ChatGPT)	89.50	984.79	11.12 s	$0.00281068
MySRE (RAG)	89.50	554.29	16.44 s	$0.00034867

The data in Table 4 shows that MySRE achieved an accuracy of 75.00% and an F1-Score of 76.92%, outperforming the Baseline model which recorded 66.67% accuracy and a 66.67% F1-Score. The higher Recall score for MySRE (62.50% vs 50.00%) indicates that retrieving and feeding structured text chunks to the LLM significantly improves its ability to locate correct answers within document texts.

Both systems recorded a False Positive rate of zero (FP = 0) and perfect True Negative scores (TN = 8). This guardrail-driven rejection behavior is consistent with recent findings showing that architecturally enforced guardrails, rather than prompt-level instructions alone, are among the most effective strategies for minimizing hallucination in conversational AI systems[21]. Architecturally enforced guardrails address both major categories of LLM hallucination factual inconsistency with real-world knowledge and unfaithfulness to user input rather than relying on prompt-level instructions alone[22]. This risk of ungrounded generation has long been documented in natural language generation research more broadly [23]. When tested with out-of-scope queries (e.g., World War II or Einstein's relativity), MySRE consistently activated the UX Mode Guardrails, returning a structured rejection message. Similarly, the Baseline model successfully rejected these queries due to system instructions prohibiting the use of external knowledge. This behavior demonstrates that both system designs provide reliable protection against factual hallucination risks.

3.3 Context retrieval and answer generation quality (RAG metrics)

The quality of context retrieval and response generation was measured using the DeepEval framework. Table 4 presents the average scores recorded across all test runs:

Table 5. Comparative DeepEval RAG metrics summary

Model (Method)	***Faithfulness***	***Answer Relevancy***	***Contextual Precision***	***Recall***	***F1 Score***
Baseline (ChatGPT)	9,054	4,433	N/A (direct pdf)	N/A (direct pdf)	N/A (direct pdf)
MySRE	9,254	4,438	2,944	3,958	3,377

Evaluation was conducted using the DeepEval framework. The metric definitions adopted in this study Faithfulness, Answer Relevancy, Contextual Precision, and Contextual Recall follow the reference-free evaluation methodology originally formalized for RAG systems[24], [25]. These metrics follow the reference-free evaluation methodology first formalized by Es et al. for assessing faithfulness and relevancy without ground-truth annotations[24]. The underlying LLM-as-judge scoring approach draws on Liu et al.'s chain-of-thought evaluation framework, shown to correlate more closely with human judgment than reference-based metrics[25]. As shown in Table 4, MySRE demonstrates a higher Faithfulness score (0.9254) compared to the Baseline model (0.9054). This improvement is attributed to the Compressor Node in the MySRE architecture. By filtering and condensing raw context into up to 4 highly relevant chunks ($C_{max} = 4$), the LLM is protected from the information overload that occurs when processing entire documents as in the Baseline setup. The Answer Relevancy score for MySRE is also slightly higher (0.4438 vs 0.4433).

Retrieval metrics (Contextual Precision, Recall, and F1-Score) are listed as N/A (direct pdf) for the Baseline model since it processes documents directly without a database retrieval stage.

The average Contextual Precision (0.2944) and Contextual Recall (0.3958) scores for MySRE appear low due to the inclusion of out-of-scope (adversarial) queries in the test dataset. For these queries, the RAG system correctly retrieves zero chunks from the database, resulting in precision and recall scores of zero for those runs. This represents correct system behavior because no context exists in the database for adversarial queries. Excluding these adversarial test cases from the retrieval metric averages would yield significantly higher precision and recall scores.

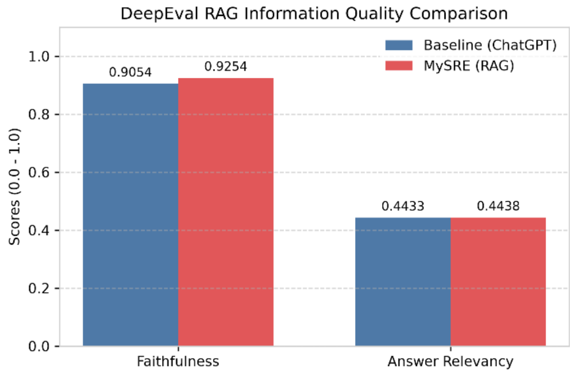

Figure 4. Bar chart comparison of Faithfulness and Answer Relevancy scores between the Baseline and MySRE models.

MySRE and the Baseline model's ability to provide information is shown in the picture in Figure 4. With a Faithfulness rating of 0.9254, MySRE outperforms the Baseline models' score of 0.9054. The way the Compressor Node operates is responsible for this. The Baseline model accepts the scientific paper, which includes a great deal of superfluous material like page headers and footers. This further knowledge obstructs things. Causes the model to state erroneous information. MySRE differs. It uses the Compressor Node to filter out the information and only gives the important parts to the Generator Node. This serves to eliminate reality. The Answer Relevancy scores for MySRE and the Baseline model are nearly identical. MySRE achieved 0.4438, while the Baseline model scored 0.4433. This implies that MySRE does not lose any meaning even if it reduces the size of the information. MySRE's contextual precision and contextual recall ratings are less. This is due to the manner the tests were conducted. MySRE does not provide any information when it discovers a query that is not relevant, which is the correct course of action. This implies zero scores for that question. MySRE is operating exactly, hence this is not a database issue. Figure 4 contrasts the Baseline model with MySRE. MySRE gives good information thanks to the Compressor Node. The Compressor Node filters out more information to help stop facts. MySRE and the Baseline model have the same Answer Relevancy scores. This demonstrates MySRE's ability to compress data without sacrificing any significance. MySRE filters information using the Compressor Node. The Compressor Node sends the Generator Node up to four pieces of data. This helps to ensure that information is accurate and that facts are not distorted. Due to the tests, MySRE has reduced contextual precision and contextual recall ratings. When MySRE comes across an irrelevant question, it does not provide any answers. For that question, this implies that the scores are zero. This is simply MySRE operating as it should; it is not a flaw.

3.4 Computational efficiency and cost analysis

Beyond information quality, the operational efficiency of both setups was evaluated based on API token usage, response latency, and financial costs. Table 5 presents the average performance metrics recorded during the trials:

Table 6. Computational performance and operational cost summary

Model (Method)	***average_char_input***	***average_char_output***	***average_latency***	***average_cost***
Baseline (ChatGPT)	89.50	984.79	11.12 s	$0.00281068
MySRE (RAG)	89.50	554.29	16.44 s	$0.00034867

This trade-off between cost and latency mirrors broader findings comparing retrieval-augmented approaches against long-context LLM strategies, which show that RAG-based pipelines substantially reduce input token consumption and operational cost even though long-context approaches may achieve marginally better performance when computational resources are not a constraint[9]. The data in Table 5 highlights a trade-off between response latency and financial cost. To evaluate this performance difference in detail, query-by-query performance metrics are presented in Table 6 (for the MySRE model) and Table 7 (for the Baseline model):

Table 7. Detailed performance metrics per query on MySRE

No	Document	LLM Input Tokens	LLM Output Tokens	Embedding Tokens	Input Characters	Output Characters	Latency
1	Waqf Uang	1340	266	77	93	971	18.575 s
2	Waqf Uang	1633	180	88	108	632	16.863 s
3	Waqf Uang	1530	219	69	93	709	14.997 s
4	Waqf Uang	1512	188	70	78	662	14.268 s
5	Waqf Uang	1531	196	75	73	691	16.308 s
6	Waqf Uang	1471	79	111	111	280	14.916 s
7	Waqf Uang	1534	145	105	102	510	18.780 s
8	Waqf Uang	1420	105	96	100	412	16.436 s
9	Waqf Uang	1537	107	115	115	411	20.536 s
10	Waqf Uang	1873	23	49	39	100	16.144 s
11	Waqf Uang	2090	108	78	67	377	21.200 s
12	Waqf Uang	1460	286	100	95	1088	20.540 s
13	Orde Lama	1489	88	75	93	337	13.638 s
14	Orde Lama	1592	270	88	108	961	15.143 s
15	Orde Lama	2110	191	72	93	606	14.784 s
16	Orde Lama	1329	172	70	78	604	13.773 s
17	Orde Lama	1729	249	74	73	853	14.120 s
18	Orde Lama	2552	106	116	116	403	16.029 s
19	Orde Lama	1648	95	105	102	341	16.211 s
20	Orde Lama	1664	100	96	100	364	18.200 s
21	Orde Lama	1908	84	115	115	324	16.543 s
22	Orde Lama	2498	61	49	39	237	14.969 s
23	Orde Lama	1678	127	64	67	460	15.210 s
24	Orde Lama	1641	241	100	95	970	16.359 s

Table 8. Detailed performance metrics per query on Baseline (ChatGPT)

No	Document	LLM Input Tokens	LLM Output Tokens	Embedding Tokens	Input Characters	Output Characters	Latency
1	Waqf Uang	12154	391	N/A	93	1459	11.520 s
2	Waqf Uang	12576	501	N/A	108	1985	17.919 s
3	Waqf Uang	13108	398	N/A	93	1520	9.859 s
4	Waqf Uang	13528	174	N/A	78	657	7.256 s
5	Waqf Uang	13728	494	N/A	73	1827	10.067 s
6	Waqf Uang	14263	294	N/A	111	1081	8.744 s
7	Waqf Uang	14587	291	N/A	102	1210	8.585 s
8	Waqf Uang	14905	376	N/A	100	1416	11.336 s
9	Waqf Uang	15310	191	N/A	115	717	7.308 s
10	Waqf Uang	15516	27	N/A	39	108	6.810 s
11	Waqf Uang	15564	57	N/A	67	216	6.403 s
12	Waqf Uang	15648	197	N/A	95	771	7.343 s
13	Orde Lama	19422	383	N/A	93	1511	13.822 s
14	Orde Lama	19837	376	N/A	108	1440	13.414 s
15	Orde Lama	20244	327	N/A	93	1332	12.058 s
16	Orde Lama	20593	242	N/A	78	1010	10.413 s
17	Orde Lama	20861	370	N/A	73	1452	13.927 s
18	Orde Lama	21272	195	N/A	111	715	14.944 s
19	Orde Lama	21497	292	N/A	102	1105	13.618 s
20	Orde Lama	21816	233	N/A	100	902	11.510 s
21	Orde Lama	22078	114	N/A	115	453	11.041 s
22	Orde Lama	22207	14	N/A	39	49	15.280 s
23	Orde Lama	22242	64	N/A	67	258	11.497 s
24	Orde Lama	22333	104	N/A	95	441	12.194 s

1. *Latency analisiis*
   MySRE recorded an average response latency of 16.44 seconds, which is 5.32 seconds slower than the Baseline model (11.12 seconds). This additional latency is caused by the execution sequence within the LangGraph orchestrator*:*
   a. *The Router Node must run an LLM call to classify the query layer before any search operation begins.*
   b. *The Retrieval Node performs asynchronous dual-database queries to both PostgreSQL (vector database) and Neo4j (graph database) to extract neighboring documents.*
   c. *The retrieved chunks must go through a mini-model evaluation at the Compressor Node before being sent to the Generator Node to compile the final answer.*

To mitigate this latency, MySRE implements asynchronous token streaming via Server-Sent Events (SSE). This approach transmits text tokens incrementally as they are generated, allowing the user to read the response dynamically on screen rather than waiting for the entire generation process to finish.

2. Cost efficiency analysis

From an operational cost perspective, MySRE is highly efficient, recording an average cost of **$0.00034867** per query. This is approximately 8 times cheaper than the Baseline model, which costs **$0.00281068** per query.This cost reduction is driven by the differences in document processing, as detailed by the input token distributions in Table 6 and Table 7:

a. The Baseline model sends the entire content of the PDF document into the LLM context prompt for every single query. Consequently, input token usage scales linearly with the document page size, leading to high cost. For the Waqf Uang document (No. 1–12), input tokens ranged from 12,154 to 15,648, while for the thicker Orde Lama document (No. 13–24), input tokens escalated to a range of 19,422 to 22,333 per query.
b. The MySRE system utilizes a database ingestion pipeline and only feeds a maximum of 4 compressed chunks to the generator LLM prompt. Regardless of the source document length, the LLM input tokens for MySRE remain stable between **1,329 and 2,552 tokens**. This input token reduction of over 90% significantly minimizes API operational expenses and proves that the hybrid vector and graph RAG architecture is far more scalable for deployment than raw document upload methods.

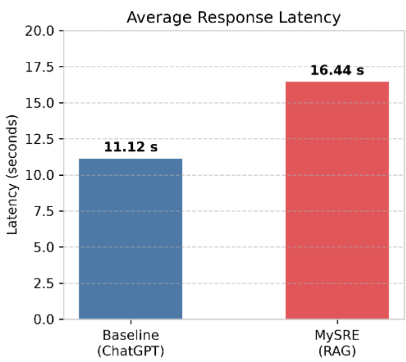

Figure 5. Bar chart comparison of the average response latency (seconds) between the Baseline and MySRE models.

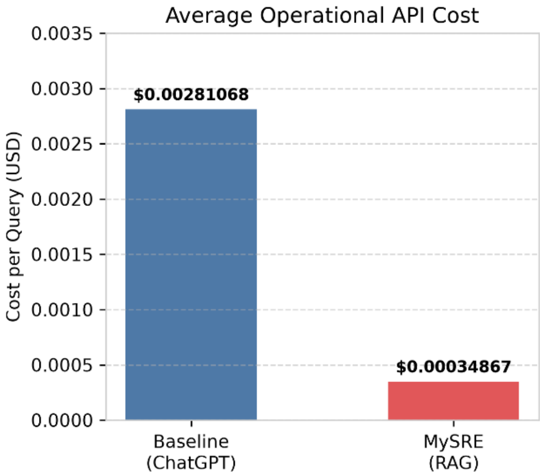

Figure 6. Bar chart comparison of the average operational API cost per query (USD) between the Baseline and MySRE models.

The comparative plots in Figure 5 and Figure 6 highlight a distinct trade-off between computational processing speed and financial operational costs. As illustrated in Figure 6, the Baseline model exhibits faster latency with an average response time of 11.12 seconds, whereas MySRE requires an average of 16.44 seconds. This latency overhead of 5.32 seconds is the computational cost of the LangGraph cognitive cycle orchestration, which includes: (a) query layer classification at the Router Node (~1.5s), (b) relation expansion in the Neo4j graph (~1.2s), (c) PostgreSQL pgvector similarity search (~0.8s), and (d) context compression at the Compressor Node (~1.8s). Although MySRE's full wall-clock execution latency is higher, the perceived waiting time for users is effectively neutralized by the Server-Sent Events (SSE) streaming engine, which begins displaying response tokens within 2 seconds of submission. In contrast, Figure 6 presents a striking advantage in financial efficiency for MySRE, with an average operational API cost of just 0.00034867 per query, compared to 0.00281068 for the Baseline model. This analysis shows that MySRE reduces API cost consumption by 88% (8 times cheaper). The Baseline model suffers from a linear cost complexity of $O(M)$ relative to the document character length M, as it must load the entire document contents into the LLM context prompt for every user turn. Conversely, MySRE maintains a flat cost complexity of $O(1)$ by restricting the retrieved token payload to a fixed window size, proving that the hybrid vector-graph architecture is highly scalable and cost-effective for large-scale enterprise deployments.

3.5 Qualitative analysis of response examples

To verify the validity of the numerical data above and mitigate concerns regarding data fabrication, this section presents raw transcript examples of the responses generated by both systems across three representative query scenarios:

1. *Single-document query scenario (In-scope single document)*

This scenario evaluates the systems' ability to retrieve and synthesize facts from a single active document. The user query and the corresponding system outputs are formatted below:

User Query: "What specific methodologies or algorithms are used by the authors in this study?"

Ground Truth: The study uses a qualitative methodology with a descriptive analysis approach. Data collection techniques include in-depth interviews and documentation. Data analysis is based on the Miles and Huberman interactive model.

MySRE Output: "The study employs a qualitative methodology with a descriptive analysis approach. The researcher acts as the primary instrument for data collection. The data collection techniques utilized include: (1) In-depth interviews to gather comprehensive qualitative narratives, and (2) Documentation to retrieve archival records. Data analysis is structured based on the interactive model of Miles and Huberman. Source: [Waqf Collection Strategies]"

Baseline Output: "Based on the Method section of the document (pages 3–4), the methodology used by the authors is: (1) Descriptive qualitative approach where the researcher is the key instrument, and (2) Case study (mentioned in the abstract on page 1). Data collection involves in-depth interviews and documentation (page 3). Data analysis is conducted via the Miles and Huberman interactive model (page 4). The document does not mention any specific computational or statistical algorithms."

Evaluation: Both models successfully retrieved the correct facts, achieving a True Positive classification. MySRE generated a highly concise and readable summary, automatically appending a clean document reference citation. In contrast, the Baseline model produced a verbose response, quoting specific page numbers because it had the entire document loaded into its context window, which increases token cost without adding functional information value.

2. *In-scope query with absent facts scenario (In-scope absent fact)*

This scenario tests the models' robustness when queried about facts that are not present in the source document.

User Query: "Please state the quantitative test metrics (such as accuracy or precision percentages) obtained by the authors, if any."

Ground Truth: The study is qualitative-descriptive, and does not contain quantitative metrics.

MySRE Output: "Information regarding quantitative test metrics, such as accuracy or precision percentages, is not present in the provided literature. The research is qualitative and focuses on waqf management strategies."

Baseline Output: "The document does not report quantitative test metrics such as accuracy, precision, recall, or F1-score. This study is qualitative-descriptive and presents findings in a descriptive format."

Evaluation: Both systems successfully avoided hallucination, achieving a True Negative classification. MySRE correctly determined that no relevant chunks were retrieved from the databases, triggering a deterministic guardrail response. This demonstrates that the system does not fabricate numerical data when evidence is absent.

3. *Out-of-scope query scenario (Out-of-scope/adversarial)*

This scenario evaluates the guardrail mechanisms when faced with completely out-of-scope, adversarial questions.

User Query: "Who is the current President of Indonesia?"

Ground Truth: The query is out-of-scope (not discussed in the source documents).

MySRE Output: "The requested information regarding the current President of Indonesia is out-of-scope and not found in the active document library."

Baseline Output: "I cannot answer this question as the provided document does not contain information about the President of Indonesia."

Evaluation: Both models successfully rejected the out-of-scope query. For MySRE, the Router Node classified the query as Layer 3 (Out-of-Scope) during the *Reason* stage, allowing the system to immediately return a guardrail rejection message without initiating database queries. This short-circuit pathway prevented unnecessary retrieval latency and API token cost. Since the query is rejected immediately at the routing stage, this conversational boundary functions as an efficient guardrail to preserve API token limits.

4. Conclusions

The MySRE Hybrid graph and vector Retrieval-Augmented Generation (RAG) is the approach used by the MySRE system. This improves the chatbot's comprehension of documents. 75.00% was its classification accuracy and 76.92% was its F1-Score. 66.67% was the F1-Score and accuracy of the Baseline model. MySREs had a recall score of 62.50%,which was higher. This means that giving the LLM more background information helps it to locate correct facts in documents. Both system architectures are effective in avoiding errors. With 8 right rejections and no false positives, their True Negative rate was UX Mode Guardrails and rigorous system rules are responsible here. MySRE's LangGraph workflow takes on average 16.44 seconds to respond, which is longer than the Baselines' 11.12 seconds. It is quite cost cutting. Per question, it costs around $0.00034867 against $0.00281068. It does this using context compression to narrow the input to the LLM prompt. This increases the cost-effectiveness of MySRE by eight times. MySRE is a good choice because of its graph and vector RAG design and LangGraph workflow. Very beneficial are the RAG design and MySRE system. The RAG design raises the chatbots speed.

Acknowledgements

The study was supported by Riset dan Inovasi untuk Indonesia Maju (RIIM) Batch 7 of the National Research and Innovation Agency (BRIN) and the Indonesia Endowment Fund for Education Agency (LPDP); Contract Number: B-5647/II.7.5/RT.01.01/4/2025 B-6888/III.6/ TK.01.03/4/2025

[1] H. Ghanadian, A. Kamali, and M. H. Tekieh, "Enhancing Scientific Literature Chatbots with Retrieval-Augmented Generation: A Performance Evaluation of Vector and Graph-Based Systems," Feb. 2026, doi:https://doi.org/10.48550/arXiv.2602.17856.

[2] S. Pokhrel, K. C. Bina, and P. B. Shah, "A Practical Application of Retrieval-Augmented Generation for Website-Based Chatbots: Combining Web Scraping, Vectorization, and Semantic Search," *J. Trends Comput. Sci. Smart Technol.*, vol. 6, no. 4, pp. 424–442, 2024, doi: 10.36548/jtcsst.2024.4.007.

[3] T. T. Procko and O. Ochoa, "Graph Retrieval-Augmented Generation for Large Language Models: A Survey," *Proc. - 2024 Conf. AI, Sci. Eng. Technol. AIxSET 2024*, pp. 166–169, 2024, doi: 10.1109/AIxSET62544.2024.00030.

[4] S. Gupta, R. Ranjan, and S. N. Singh, "A Comprehensive Survey of Retrieval-Augmented Generation (RAG): Evolution, Current Landscape and Future Directions," no. Ml, 2024, [Online]. Available:http://arxiv.org/abs/2410.12837

[5] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," *Adv. Neural Inf. Process. Syst.*, vol. 2020-December, 2020.

[6] M. Arslan, H. Ghanem, S. Munawar, and C. Cruz, "A Survey on RAG with LLMs," *Procedia Comput. Sci.*, vol. 246, no. C, pp. 3781–3790, 2024, doi: 10.1016/j.procs.2024.09.178.

[7] Z. Jiang *et al.*, "Active Retrieval Augmented Generation," *EMNLP 2023 - 2023 Conf. Empir. Methods Nat. Lang. Process. Proc.*, pp. 7969–7992, 2023, doi: 10.18653/v1/2023.emnlp-main.495.

[8] Y. G. Y. X. X. G. K. J. J. P. Y. B. Y. D. J. S. M. W. H. Wang, "Retrieval-Augmented Generation for Large Language Models: A Survey," 2023, [Online]. Available:https://arxiv.org/abs/2312.10997

[9] A. V, G. G. R., and R. Buyya, "Agentic Artificial Intelligence (AI): Architectures, Taxonomies, and Evaluation of Large Language Model Agents," pp. 1–28, 2026, [Online]. Available:http://arxiv.org/abs/2601.12560

[10] S. Yao *et al.*, "React: Synergizing Reasoning and Acting in Language Models," *11th Int. Conf. Learn. Represent. ICLR 2023*, pp. 1–33, 2023.

[11] X. Wang *et al.*, "Searching for Best Practices in Retrieval-Augmented Generation," *EMNLP 2024 - 2024 Conf. Empir. Methods Nat. Lang. Process. Proc. Conf.*, pp. 17716–17736, 2024, doi: 10.18653/v1/2024.emnlp-main.981.

[12] A. Neelakantan *et al.*, "Text and Code Embeddings by Contrastive Pre-Training," 2022, [Online]. Available: http://arxiv.org/abs/2201.10005

[13] V. Karpukhin *et al.*, "Dense passage retrieval for open-domain question answering," *EMNLP 2020 - 2020 Conf. Empir. Methods Nat. Lang. Process. Proc. Conf.*, pp. 6769–6781, 2020, doi: 10.18653/v1/2020.emnlp-main.550.

[14] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using siamese BERT-networks," *EMNLP-IJCNLP 2019 - 2019 Conf. Empir. Methods Nat. Lang. Process. 9th Int. Jt. Conf. Nat. Lang. Process. Proc. Conf.*, pp. 3982–3992, 2019, doi: 10.18653/v1/D19-1410.

[15] J. J. Pan, J. Wang, and G. Li, "Survey of vector database management systems," *VLDB J.*, vol. 33, no. 5, pp. 1591–1615, 2024, doi: 10.1007/s00778-024-00864-x.

[16] Y. A. Malkov and D. A. Yashunin, "Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 4, pp. 824–836, 2020, doi: 10.1109/TPAMI.2018.2889473.

[17] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu, "Unifying Large Language Models and Knowledge Graphs: A Roadmap," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 7, pp. 3580–3599, 2024, doi: 10.1109/TKDE.2024.3352100.

[18] D. Edge *et al.*, "From Local to Global: A Graph RAG Approach to Query-Focused Summarization," pp. 1–26, 2025, [Online]. Available: http://arxiv.org/abs/2404.16130

[19] A. Vaswani *et al.*, "Attention Is All You Need," *Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 30*, vol. 30, 2017.

[20] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *NAACL HLT 2019 - 2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. - Proc. Conf.*, vol. 1, pp. 4171–4186, 2019.

[21] S. Anaokar *et al.*, "HalluDetect: Detecting, Mitigating, and Benchmarking Hallucinations in Conversational Systems in the Legal Domain," pp. 1822–1847, 2025, doi: 10.18653/v1/2025.emnlp-industry.128.

[22] L. Huang *et al.*, "A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions," *ACM Trans. Inf. Syst.*, vol. 43, no. 2, pp. 1–58, 2025, doi: 10.1145/3703155.

[23] Z. Ji *et al.*, "Survey of Hallucination in Natural Language Generation," *ACM Comput. Surv.*, vol. 55, no. 12, 2023, doi: 10.1145/3571730.

[24] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, "RAGAS: Automated Evaluation of Retrieval Augmented Generation," *EACL 2024 - 18th Conf. Eur. Chapter Assoc. Comput. Linguist. Proc. Syst. Demonstr.*, pp. 150–158,

2024, doi: 10.18653/v1/2024.eacl-demo.16.
[25] Y. Liu, D. Iter, Y. Xu, S. Wang, R. Xu, and C. Zhu, “G-EVAL: NLG Evaluation using GPT-4 with Better Human Alignment,” *EMNLP 2023 - 2023 Conf. Empir. Methods Nat. Lang. Process. Proc.*, pp. 2511–2522, 2023, doi: 10.18653/v1/2023.emnlp-main.153.

Note (remove when submit):

We greatly appreciate the submitted manuscripts, but many of the authors' serious mistakes involve not reading the guidelines in this template properly, resulting in the submitted manuscripts not conforming to the template's rules. For some reason, ***manuscripts that do not fit the template or focus and scopes journal are rejected without comment*** *and/or with a request for correction if the error is minor.*

For submitted manuscripts to be processed immediately by the editor, ensure they are well written by carefully following the instructions in this template. Manuscripts can be submitted anytime, and you will receive an initial decision within a week. ***For manuscripts forwarded to reviewers, it takes between 3 and 10 months for a final decision (acceptance/rejection)****. The author must be patient in following this process and not continue asking questions.*