Dynamic Query Handling with RAG Fusion for PDF-Based Knowledge Retrieval Systems

Aakash Sonkar
Computer Science & Engineering
Government Engineering College
Raipur, India
aakashsonkar@gmail.com

Sumeet Pratap Singh
Computer Science & Engineering
Government Engineering College
Raipur, India
sumeetpratapsingh31@gmail.com

Komendra Sahu
Computer Science Engineering
Government Engineering College
Raipur, India
sahukomendra721@gmail.com

Aayush Sahu
Computer Science Engineering
Government Engineering College
Raipur, India
sahuaayush6266@gmail.com

***Abstract*—This study explores advancements in Retrieval-Augmented Generation (RAG) systems tailored for PDF-based question answering. By leveraging domain-specific PDFs as input data, the proposed framework incorporates advanced retrieval techniques, including multi-query generation and RAG fusion, to enhance generated responses' relevance and contextual accuracy. Utilizing LangChain to orchestrate interactions between the language model and vector database (Chroma DB), the system effectively processes, embeds, and retrieves information from large-scale PDF collections. Chroma DB ensures efficient storage and similarity searches for vectorized document embeddings. The integration of advanced retrieval strategies, coupled with the dynamic query-handling capabilities of LangChain, significantly improves precision, reduces hallucinations, and enriches response quality for knowledge-intensive tasks. Preliminary results demonstrate the effectiveness of this approach in delivering timely and contextually accurate answers, showcasing its potential in addressing the challenges of traditional LLMs.**

Keywords*—Retrieval-augmented generation (RAG), generative ai, chatbot, LLMs, LangChain, vector embedding, prompt template, vector database, stream lit.

I. INTRODUCTION

The rise of Large Language Models (LLMs) has transformed the field of natural language processing (NLP), enabling tasks such as question answering, summarization, and content generation with remarkable precision. LLMs like Mistral and LLaMA represent state-of-the-art architectures capable of understanding and generating human-like text. However, the effectiveness of these models depends significantly on how they process and retrieve contextual information [1].

Retrieval-augmented generation (RAG) presents a powerful solution to this problem. RAG systems retrieve information from external data sources, such as PDFs, databases, or websites, grounding the generated content in accurate and current data. This makes them particularly effective for knowledge-intensive tasks [2]. By informing LLMs of reference materials related to questions in advance, RAG reduces hallucinations and generates more accurate and reliable answers. As a result, RAG architecture resolves the information shortage problem of LLMs, providing high-quality responses without requiring additional data training [3]. To implement efficient retrieval mechanisms in RAG pipelines, tools like FAISS (Facebook AI Similarity Search) are employed. FAISS facilitates storing and rapidly retrieving large document embeddings, enabling systems to handle massive datasets without compromising performance. Advanced configurations, such as RAG Fusion and RAG Multi-query [4] further enhance retrieval

II. LITERATURE WORK

Jiang et. al. [6] presents the ARAG framework, which interleaves retrieval and generation to improve adaptability and efficiency. FLARE, a method within this framework, optimizes retrieval for long-form and complex queries using explicit instructions and confidence-based strategies. It addresses issues like redundancy and inaccuracies in text generation.

Jégou et. al. [27] presents the FAISS library's efficiency in handling large-scale, high-dimensional vector data for similarity search and clustering. It supports various similarity metrics and manages data beyond RAM capacity, making it a powerful tool for document storage and retrieval.

Z. Li et. al. [17] Proposes DMQR to enhance RAG by addressing noisy queries through four rewriting strategies. An adaptive method balances the number of rewrites with performance, significantly improving retrieval and response accuracy in academic and industry applications.

Rackauckas [28] describes about RAG-Fusion which integrates retrieval evidence into generation to improve factuality and reduce hallucinations. This approach excels in domains needing precise knowledge, such as customer support, outperforming traditional RAG models.

Hu, Zhibo et. al. [29] introduces Gradient-Guided Prompt Perturbation (GGPP) to manipulate RAG models by altering prompts. GGPP exposes vulnerabilities by prioritizing incorrect retrievals, advancing adversarial prompt methodologies, and highlighting potential risks in RAG systems.

III. METHODOLOGY

Large Language Models (LLMs), such as OpenAI's GPT and Google's BERT, are based on machine learning and deep learning technologies [5] and are large-scale models designed for training on vast amounts of data. These models serve as foundational tools in Natural Language Processing (NLP) and Natural Language Generation (NLG) tasks, where they demonstrate remarkable capabilities in understanding and generating human language. LLMs are typically trained on massive corpora of text, including books, web pages[6], and social media conversations, enabling them to handle a wide range of language-based tasks. However, LLMs encounter challenges when dealing with domain-specific or highly specialized queries[8] that require information outside their training data. LLMs in RAG models can be fine-tuned for specific tasks or domains (e.g., legal, medical, or technical domains) by training them on domain-specific datasets[13]. This helps the model generate more accurate and relevant responses when combined with the retrieval mechanism, which fetches data from the specific domain's knowledge base.

A. Retrieval Augmented Generation (RAG)

Retrieval-augmented generation (RAG) is an advanced framework that combines the power of information retrieval and text generation[9] to enhance the capabilities of language models. In RAG, a retrieval mechanism is used to fetch relevant information from external data sources, such as databases or documents, which is then passed to a language model (LLM) to generate accurate, contextually grounded, and informative responses. This approach improves the factual accuracy and relevance of generated content, particularly for complex or domain-specific queries [10].

B. RAG Model Architecture: The RAG Architecture is divided into three components:

1. Indexing
2. Retrieval
3. Generation

In the first step, RAG uses PDFs as an external knowledge base and divides it into smaller textural chunks which are further embedded and stored in a vector database. In the second step, a query is given to RAG and this query is also divided into smaller textual chunks get converted into embeddings and a similarity search is executed to compare between query's embeddings and pre-existing embeddings in the vector database. The chunks with the most similar embeddings are retrieved and given to LLM with the asked query [12][13]. In the third step, the LLM refers to these chunks for providing answers to the query asked. Refer the Fig 1. for an overview of the pipeline.

A.1 Indexing:

A.1.1 Source Data Collection and Preparation: The process of building a tailored knowledge base for a language model using PDFs begins with collecting domain-specific PDFs, such as research paper, manuals, or technical guides, which form the foundation for retrieval. These PDFs are organized into a structured directory and processed using tools like PyPDF2 or UnstructuredPDFLoader to extract text.

A.1.2 Chunking: The first step in preparing the data involves dividing the source material into smaller, manageable chunks. These chunks typically consist of sentences or paragraphs, making it easier to search and retrieve specific pieces of information[15]. By breaking down the data into smaller units, the system can more efficiently handle and utilize the information during the retrieval process. Chunking breaks the pdf's text into smaller, manageable segments to preserve context and fit model input limits.

For document D, it gets split into chunks C

$$C_1, C_2, C_3 \dots . C_k$$

such that,

$$Chunk\ Size \leq Model\ Limit$$

Overlap 'O' is maintained to preserve continuity

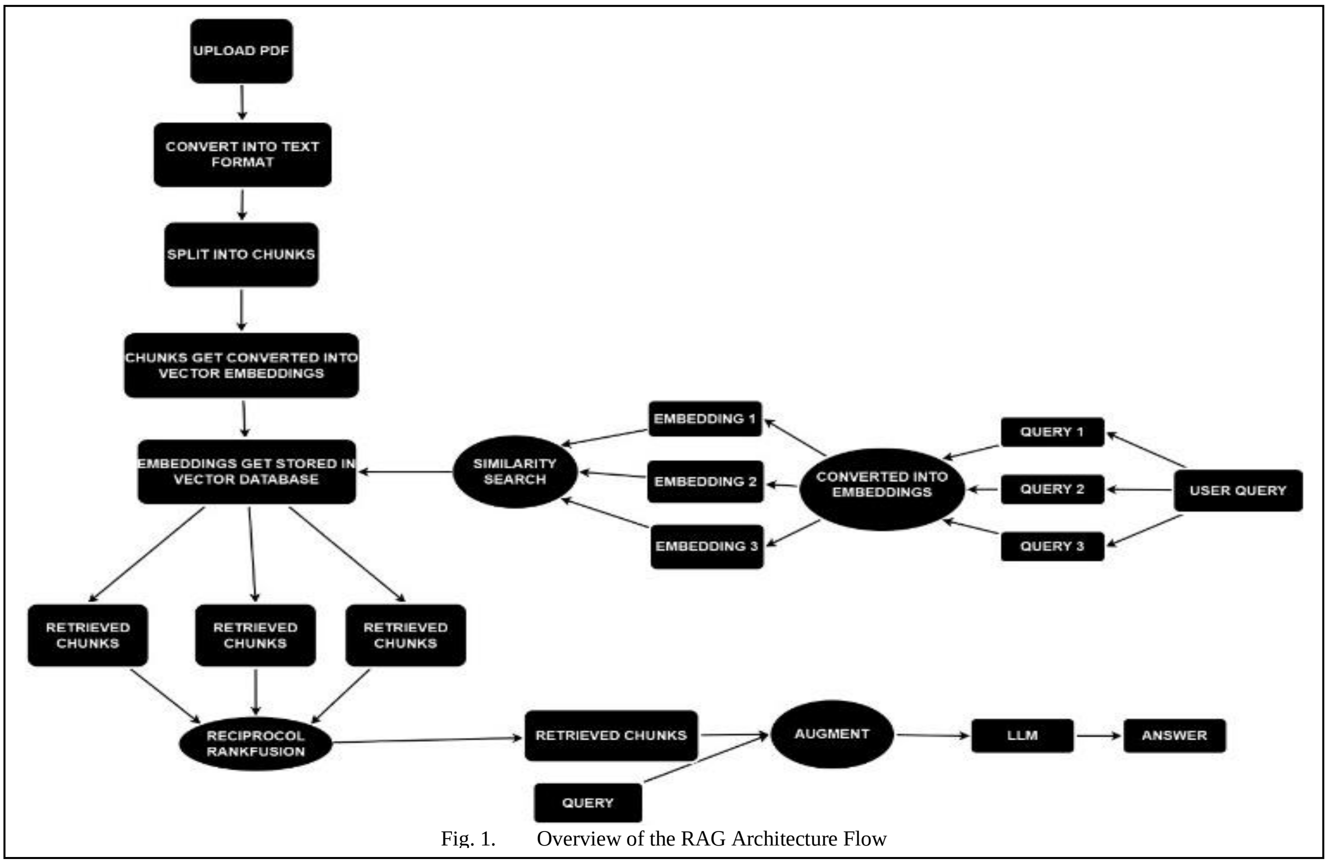

Fig. 1. Overview of the RAG Architecture Flow

$$O = Overlap\ Tokens(eg.50)$$

A.1.3 Chunking: Each text chunk is converted into a high-dimensional vector representation using embedding models (e.g., Nomic). These embeddings capture the semantic meaning of the text, making similarity searches more precise.[27] Each chunk C_i is converted to vector v_i

$$v_i = f(C_i)$$

Where:

f: Embedding function (eg. transformer model)

v_i: Resulting vector for chunk C_i

A.1.4 Constructing a Vector Database: After chunking the data and transforming it using the Nomic Embed Text model, the resulting vector embeddings are stored in a vector database such as Chroma DB.[24]This database is optimized for fast and efficient similarity searches, enabling the retrieval of contextually relevant chunks when a query is issued. The indexed structure of the database ensures that the system can quickly find and return the most relevant information based on semantic similarity.

A.2 Information Retrieval: Information Retrieval focuses on finding relevant data from a large corpus or database. In RAG, it ensures that the retrieved content aligns with the user's query, forming the basis for the generative response. When a query is entered into the system, it is transformed into vector embedding using the same model employed for the documents in the vector store.

The Retriever Rag fusion [17] component then searches for the vector database, such as Chroma DB, to identify and retrieve the most relevant chunks of information. This process ensures that the system relies on the most pertinent and up-to-date data to generate accurate and contextually grounded responses.

A.2.1 User Query Vectorization: Converts the user input into a vector representation for semantic comparison

$$v_q = f(Q)$$

Where:

f: embedding model

Q: user query which is converter to vector v_q

A.2.2 Vector Database: The database performs similarity search to find the most relevant text embeddings based on the query embedding, ensuring that only the most contextually appropriate chunks are retrieved for the generation process.

$$\text{Retriever Top-k}$$
$$\arg\max_{i\in\{1,2,\ldots\ldots.n\}} \text{Cosine Similarity}(v_q,v_i)$$

Sparse Retrieval: Generative Based on term-matching methods like TF-IDF or BM25[20]. Sparse retrieval relies on the exact word matches between the query and documents.

Formula for TF-IDF

$$TF-IDF(t,d) = TF(t,d).log(\frac{N}{DF(t)})$$

Where:

t:term

d:document

N:total number of documents

DF(t): number of documents containing term t

Dense Retrieval*:* Uses vector representations of text (embeddings) for similarity matching. Dense retrieval is more semantic, using cosine similarity:

$$CoSine\ Similarity(q,d) = q.\frac{d}{||q||||d||}$$

Where:

q: query vector

d: document vector

||q||, ||d||: magnitudes of the vectors

Advance Retrieval Techniques:

RAG-Fusion - RAG-Fusion stands as a innovative methodology within the realm of AI chatbots, enhancing the default Retrieval-Augmented Generation (RAG) by introducing additional steps aimed at refining the quality of the generated text. Refer the Fig. 2 for seeing the overview of rag fusion. This method leverages the power of multi-query generation and result reranking to address some of the inherent limitations present in traditional RAG systems[16]. Integrating these features ensures that responses generated by AI chatbots are not only more relevant but also of higher quality, catering to the dynamic needs of users seeking accurate and comprehensive information.

Multi-Query Retriever: Multiple Query Retrieval[17] in Retrieval-Augmented Generation (RAG) is an advanced mechanism designed to refine the retrieval process by generating multiple variations of a query [27]. This approach broadens the scope of search results and ensures the system identifies a diverse and comprehensive set of relevant information. It is particularly useful for queries that are complex, ambiguous, or multi-dimensional.

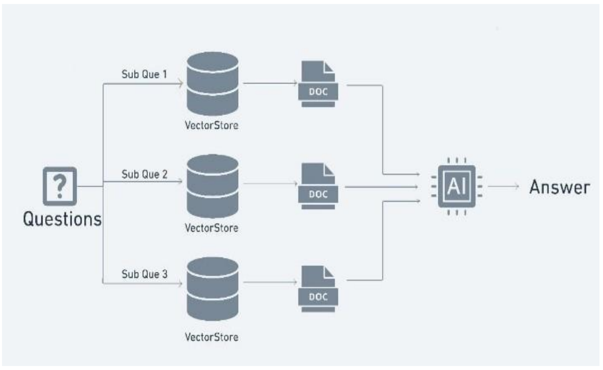

Fig. 2. Multi-Query Retrieval in RAG

Role of Vector Database The role of a vector database in a RAG system is to store vector embeddings of data chunks with metadata[19], perform efficient semantic similarity searches, handle large-scale data with advanced indexing techniques, and enable real-time, accurate retrieval for interactive applications.

A.3 Generation:

Integrating with LLM: LLMs process augmented prompts combining user queries and retrieved context.

Augmented Input: prompt = Concatenate (Query, RetrievedChunks)

Prompt Formatting: After the similarity search retrieves the most relevant text embeddings, these chunks are integrated with the user query into a structured prompt. The prompt typically follows a predefined template to guide the LLM in generating a response.

Input to the LLM*:* The formatted prompt along with augmented input is passed to the LLM for processing. The LLM uses its pre-trained knowledge combined with the provided context to generate a coherent and contextually accurate response.

Response Generation: The LLM produces a natural language response[18] that aligns with the user query and the retrieved context.

The response is formulated to be user-facing, addressing the question or providing the requested information.

B. Advantages of RAG Architecture:

Accuracy: Dynamically retrieves up-to-date and relevant information, reducing the risk of hallucinations.

Scalability: Efficiently handles large-scale data with tools like Chroma DB and advanced embedding techniques.

Versatility: Adapts to various domains and applications, offering tailored, context-rich responses.

Efficiency: Combines lightweight retrieval mechanisms with powerful LLMs, optimizing resource usage.

The RAG model framework introduces a paradigm shift in Generative AI by creating glass-box models. It greatly enhanced the ability of generative models to provide accurate information, especially in knowledge-intensive domains. This integration has become the backbone of many advanced NLP applications, such as chatbots, virtual assistants, and automated customer service systems

In this specific RAG-based project, LangChain plays a critical role in the following ways:

1)Connecting the Vector Store (Chroma DB) with LLMs: LangChain facilitates easy retrieval of semantic data from Chroma DB based on user queries. It then constructs prompts that incorporate both the query and the retrieved information, allowing the LLM to generate responses grounded in external knowledge. 2)Dynamic Prompting: It ensures that the retrieved documents are integrated effectively with the user query to form a prompt that is both relevant and contextually appropriate for the LLM. This improves the quality of the generated answers. 3)Managing Multiple Retrieval and Generation Steps: LangChain orchestrates the multiple stages of RAG—retrieving relevant data from the vector store, dynamically generating the query prompt, passing it to the LLM, and then generating and formatting the final output for the user[21].

A.1 Algorithm 1: Load and Preprocess PDF: This step involves loading a PDF document from a specified path and extracting its contents into a structured format. Preprocessing may include cleaning the text, removing unnecessary characters, and standardizing the format to ensure compatibility with subsequent steps.

```
1.Check if local_path is not empty
  1.1 If local_path exists then
    1.1.1 Initiate loader as UnstructuredPDFLoader (file_path=local_path)
    1.1.2 Load the data using loader.load()
    1.1.3 Print "PDF loaded successfully"
  1.2 else print "Upload a PDF file"
2. Return data
```

A.2 Algorithm 2: Text Splitting: This algorithm divides the loaded text into smaller, manageable chunks based on a predefined size (chunk size) and overlap (chunk overlap). The overlap ensures context continuity between chunks, which is critical for maintaining coherence in downstream processes like embedding generation or question answering.

```
1. Initialize text splitter with c = 7500 and o = 100
2. Split data into chunks using text splitter. split_documents(data)
3. Return chunks
```

*A.3 Algorithm 3: Build Vector Database :*This step involves encoding the text chunks into numerical vectors using an embedding model (E) and storing them in a vector database[25]. The database enables efficient similarity searches and retrieval tasks, which are essential for augmenting user queries with relevant context.

```
1 Initialize E as OllamaEmbeddings(model="nomic-embed-text", show_progress=True)
2 Create vector_db using Croma.from_documents (documents= chunks,embedding = E,collection_name ="local-rag").
3 Return vector_db
```

A.4 Algorithm 4: Query Augmentation: This algorithm enhances the user's query by generating alternative variations or rephrasings. It uses a language model and a prompt template to produce queries that better align with the retriever's indexing, improving retrieval accuracy and relevance.

```
1 Define QUERY_PROMPT=Generate five alternative versions of the user question for retrieval
2 Use Multi Query Retriever. from_llm (vector_db.as_retriever(), llm,prompt = QUERY_PROMPT)
3 Generate alternative queries alt_queries
4 Return alt_queries
```

.

A.5 Algorithm 5: Retrieval-Augmented Generation (RAG) Chain: This method combines information retrieval with generative language models[19]. The retriever fetches relevant data chunks based on a query, and the language model generates a response by incorporating the retrieved context. This hybrid approach ensures accurate and context-aware answers[24].

```
1 Define RAG_PROMPT as
Answer the question based ONLY on the following context-contextQuestion:question
2 Initialize chain with compnents:
2.1 Pass retriever as "context"
```

```
2.2 Use RunnablePassthrough() for "question"
2.3 Use prompt for formatting the context and question
2.4 Use llm to generate the response
2.5 Parse the response using StrOutputParser()
3 Return y
```

TABLE I. COMPARISION OF FINE TUNED RAG & BASE MODELS

Factors	**Fine-Tuning**	**RAG**	**Base Model**
Nature of the Task	Highly specialized tasks, domain-specific language.	Dynamic tasks needing real-time information retrieval	General tasks, prototyping, broad applicability
Data	Static	External knowledge bases	Not Specialized
Resource Constraints	High	High	Low demand

Table 1. shows a comparison between three methods: Fine-Tuning, RAG (Retrieval-Augmented Generation), and Base Models, based on several key factors. Fine tuning is most effective for highly specialized tasks that require domain-specific language, relying on static or proprietary data. It demands high computational resources for training. In contrast, RAG is suited for dynamic tasks that require real-time information retrieval, depending on access to up-to-date or external large knowledge bases. However, it comes with higher inference costs and infrastructure complexity [11]. Base Models, on the other hand, are ideal for general tasks, prototyping, and applications with broad use cases. They do not require specialized or up-to-date information and are low in resource demand, making them quick to deploy.

IV. RESULTS & DISCUSSION

TABLE II. BLEU,ROGUE-1,BERT P,BERT R,BERT F1 ,DIVERSITY SCORESE OF DIFFERENT OLLAMA MODELS

Model	**BLEU**	**ROUGE-1**	**BERT P**	**BERT R**	**BERT F1**	**PERPLEXITY**	**DIVERSITY**
MISTRAL	0.041	0.164	0.723	0.631	0.674	24.537	0.937
LLAMA 3.1	5.970	0.068	0.637	0.539	0.584	27.999	0.920
GEMMA	3.290	0.053	0.684	0.557	0.614	27.932	0.963
PHI3.5	0.285	0.273	0.633	0.624	0.629	63.435	0.962
QWEN2	0.559	0.239	0.766	0.669	0.715	39.471	0.981

In Table II. the evaluation of several language models—Qwen 2, Phi 3.5, Gemma, Mistral, and LLaMA 3.1—was conducted across multiple performance metrics, including BLEU, ROUGE-1, BERT scores, perplexity, and diversity. The BLEU score, which measures the precision of n-grams between the generated responses and reference texts, was calculated using the sacrebleu library, with Qwen 2 achieving the highest BLEU score of 0.5594, indicating superior surface-level accuracy.

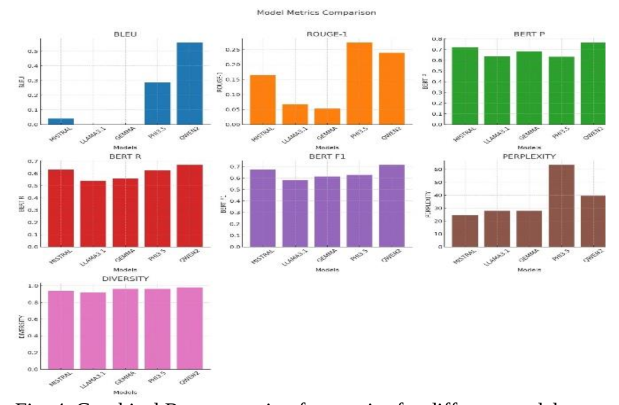

Fig. 4. Graphical Representation for metrics for different models

The ROUGE-1 score, reflecting the overlap of unigrams between generated and reference responses, was assessed using the rouge score library, with Phi 3.5 performing well at 0.2735, highlighting its strength in surface-level matching but also revealing challenges in fluency due to its high perplexity. BERT scores, evaluating the alignment of generated and reference texts at the semantic level, were computed using the Bert score library, with Qwen 2 also excelling in BERT Precision (0.7663), indicating strong semantic alignment. Perplexity, an indicator of fluency, was computed using GPT-2's log-likelihood estimation, with Mistral showing the lowest perplexity, reflecting its fluency and coherence in text generation. Lastly, diversity was measured by analyzing the uniqueness of bigrams in the generated text, with Qwen 2 achieving a high diversity score of 0.9815, suggesting a well-balanced ability to generate varied and creative responses. Overall, Qwen 2 demonstrated a well-rounded performance across all metrics, emerging as the top performer, while other models such as Phi 3.5, Gemma, Mistral, and LLaMA 3.1 each exhibited strengths in specific areas, such as diversity, fluency, and semantic precision, depending on the use case.

V. CONCLUSION

Retrieval-Augmented Generation (RAG) Fusion presents a transformative solution to key challenges in AI-driven information systems, addressing issues such as hallucination, outdated knowledge, and response accuracy by grounding outputs in reliable retrieved data. Its modular architecture offers flexibility and ease of integration, while leveraging local LLMs ensures data security and privacy, making it highly suitable for sensitive and enterprise-level applications. Beyond its current capabilities, future advancements in RAG systems open several promising directions.

Vision-Based RAG can extend the system's capabilities by integrating image and video analysis, enabling holistic comprehension of documents containing visual elements like graphs and charts, which is critical for interpreting complex content. Additionally, interactive and context-aware multi-query generation can refine alternative queries dynamically in real-time through user interaction, improving the system's ability to manage complex, nuanced multi-turn dialogues. Real-time document updating further enhances adaptability, allowing continuous updates to the vector database without requiring retraining, which is invaluable in dynamic environments such as news updates or legal proceedings. Scalability and efficiency improvements will optimize RAG systems to manage vast document repositories with faster retrieval times and reduced latency, catering to enterprise and academic use cases. Finally, advancements in deep learning techniques, including transformer models, promise to improve contextual understanding and reduce noise, delivering precise and accurate answers even in highly technical domains.

These developments collectively position RAG Fusion as a robust framework capable of addressing increasingly complex real-world challenges in a secure, efficient, and scalable manner.

REFERENCES

[1] C. Jeong, "Implementation of generative AI service using LLM application architecture: based on RAG model and LangChain framework," J. Intell. Inform. Syst., vol. 29, no. 4, pp. 129-164, Dec. 2023.

[2] A. A. Khan, K. K. Kemell, M. T. Hasan, J. Rasku, and P. Abrahamsson, "Developing Retrieval Augmented Generation (RAG) based LLM Systems from PDFs: An Experience Report," arXiv preprint arXiv:2410.15944v1, Oct. 2024.

[3] M. Shanahan, "Talking about Large Language Models," Communications of the ACM, vol. 67, no. 2, pp. 68-79, 2024.

[4] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang. 2023. Retrieval-Augmented Generation for Large Language Models: A Survey.arXiv preprint arXiv:2312.10997 (2023).

[5] Q. Ai, T. Bai, Z. Cao, Y. Chang, J. Chen, Z. Chen, Z. Cheng, S. Dong, Z. Dou, F. Feng, S. Gao, J. Guo, X. He, Y. Lan, C. Li, Y. Liu, Z. Lyu, W. Ma, J. Ma, and Z. Ren. 2023. Information Retrieval meets Large Language Models: A strategic report from Chinese IR community. FundamentalResearch4(2023),80

[6] Z. Jiang, F. F. Xu, L. Gao, Z. Sun, Q. Liu, J. Dwivedi-Yu, Y. Yang, J. Callan, and G. Neubig, "Active Retrieval Augmented Generation," in Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, pp. 7969-7992, 2023.

[7] S. Siriwardhana, R. Weerasekera, E. Wen, T. Kaluarachchi, R. Rana, and S. Nanayakkara. 2022. Improving the Domain Adaptation of Retrieval Augmented Generation (RAG) Models for Open Domain Question Answering. Electronics 11, 20 (2022), 3388.

[8] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," Facebook AI Research, University College London, New York University

[9] Monteiro, H. (2024). Chatting Over Course Material: The Role of Retrieval Augmented Generation Systems in Enhancing Academic Chatbots. Luleå University of Technology.

[10] S. Barnett, S. Kurniawan, S. Thudumu, Z. Brannelly, and M. Abdelrazek, "Seven Failure Points When Engineering a Retrieval Augmented Generation System," in 2024 IEEE/ACM 3rd International Conference on AI Engineering – Software Engineering for AI (CAIN), Geelong, Australia, 2024.

[11] X. Zheng, F. Che, J. Wu, S. Zhang, S. Nie, K. Liu, and J. Tao, "KS-LLM: Knowledge Selection of Large Language Models with Evidence Document for Question Answering," arXiv preprint arXiv:2404.15660v1, 2024

[12] Shervin Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large Language Models: A Survey," arXiv preprint arXiv:2402.06196v2, Feb. 20, 2024

[13] S. Veturi, S. Vaichal, R. L. Jagadheesh, N. I. Tripto, and N. Yan, "RAG based Question-Answering for Contextual Response Prediction

[14] H. Koo, M. Kim, and S. J. Hwang, "Optimizing Query Generation for Enhanced Document Retrieval in RAG," arXiv preprint arXiv:2407.12325v1, 2024.

[15] Z. Rackauckas, A. Câmara, and J. Zavrel, "Evaluating RAG-Fusion with RAGElo: an Automated Elo-based Framework," arXiv preprint arXiv:2406.14783v2, 2024.

[16] Z. Rackauckas, "RAG-FUSION: A New Take on Retrieval-Augmented Generation," arXiv preprint arXiv:2402.03367v2, 2024.

[17] Z. Li, J. Wang, Z. Jiang, H. Mao, Z. Chen, J. Du, Y. Zhang, F. Zhang, D. Zhang, and Y. Liu, "DMQR-RAG: Diverse Multi-Query Rewriting for Retrieval-Augmented Generation," arXiv preprint arXiv:2411.13154v1, 2024.

[18] X. Liu, J. Wang, J. Sun, X. Yuan, G. Dong, P. Di, W. Wang, and D. Wang, "Prompting Frameworks for Large Language Models: A Survey,"arXivpreprintarXiv:2311.12785v1,2023.

[19] T. Taipalus, "Vector database management systems: Fundamental concepts, use-cases, and current challenges," Cognitive Systems Research, vol. 77, pp. 35-47, 2023.

[20] S. Seyed Monir, I. Lau, S. Yang, and D. Zhao, "VectorSearch: Enhancing document retrieval with semantic embeddings and optimized search," arXiv preprint arXiv:2409.17383v1, 2024.

[21] Huang, J.-T., Sharma, A., Sun, S., Xia, L., Zhang, D., Pronin, P., Padmanabhan, J., Ottaviano, G., and Yang, L., "Embedding-based retrieval in Facebook search," in Proc. 26th ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining, 2020, pp. 2553–2561.

[22] Z. Jing, Y. Su, and Y. Han, "When large language models meet vector databases: A survey," arXiv preprint arXiv:2402.01763v3, 2024.

[23] S. Jeong, J. Baek, S. Cho, S. J. Hwang, and J. C. Park, "Adaptive-RAG: Learning to adapt retrieval-augmented," arXiv preprint arXiv:2403.14403v2, 2024.

[24] OpenAI, "GPT-4 technical report," arXiv preprint arXiv:2303.08774, 2023.

[25] Augmented Generation (RAG) Models for Open Domain Question Answering.Electronics11,20(2022),3388.

[26] S. Vuković, M. Kostić, and D. Mugoša, "Proposal for Enhancing Legal Advisory Services in the Montenegrin Banking Sector with Artificial Intelligence," 2024 16th International Conference on Artificial Intelligence and Law (ICAIL), pp. 1-8, IEEE, 2024.

[27] Jégou, Hervé, et al. "Faiss: Similarity search and clustering of dense vectors library." Astrophysics Source Code Library (2022): ascl-2210.

[28] Rackauckas, Zackary. "Rag-fusion: a new take on retrieval-augmented generation." arXiv preprint arXiv:2402.03367 (2024).

[29] Hu, Zhibo, et al. "Prompt perturbation in retrieval-augmented generation based large language models." Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. 2024.