ISNNI: 3007-3391

HEC Category: Y

Journal of Computers and Intelligent Systems

Vol. 3, No. 4, 2025, pages 283 - 296

Journal Homepage:

https://journals.iub.edu.pk/index.php/JCIS/

Agentic Retrieval Augmented Generation: A Compact Review

Sagar Chhabriya[1], Sumra Khan[2], Dr. Rizwan Khan[3], Mansoor Ahmed[4]

sagarchhabriya.bscskdkf22@iba-suk.edu.pk [1]Department of Computer Science, Sukkur IBA University, Sukkur, Pakistan

sumra.khan@shu.edu.pk [2]Department of Computer Science, Salim Habib University, Karachi, Pakistan

rizwankhan@iba.edu.pk [3]Department of Computer Science, IBA Karachi, Karachi, Pakistan

mansoorr.email@gmail.com [4]Department of Computer Science, Sukkur IBA University, Sukkur, Pakistan

ARTICLE INFO

Article History:

Received:	October 15, 2025
Revised:	November 20, 2025
Accepted:	December 25, 2025
Available Online:	December 25, 2023

Keywords:

Retrieval Augmented Generation (RAG)
Autonomous Agent Architectures Dynamic Knowledge Integration
Multi-Step Reasoning

Classification Codes:

Funding:

This research received no specific grant from any funding agency in the public or not-for-profit sector.

ABSTRACT

Retrieval-Augmented Generation (RAG) enhances large language models (LLMs) by incorporating external knowledge during inference, improving factual accuracy and contextual relevance. However, conventional RAG systems often struggle with complex, dynamic, or multi-step tasks due to their static retrieval-generation pipelines. Agentic RAG addresses these limitations by embedding autonomous agents into the RAG framework, enabling capabilities such as planning, tool use, memory integration, and adaptive decision-making. This review outlines the architecture of Agentic RAG, highlights its advantages in dynamic applications like long-form question answering and research automation, and discusses challenges such as latency, control, and ethical design.

Corresponding Author's Email: sagarchhabriya.bscskdkf22@iba-suk.edu.pk

Citation:

1. Introduction

Large Language Models (LLMs) [1] have revolutionized natural language processing by demonstrating exceptional capabilities in understanding and generating text that is human-like. Despite their remarkable performance, these models are inherently limited by the static nature of their training data, which can lead to outdated knowledge and an increased risk of generating hallucinated or unverifiable information. To address this limitation, RAG [2] was introduced as a hybrid approach that augments LLMs with dynamic access to external knowledge sources during inference. With the generative ability of LLMs and the factual content presented by content retriever, RAG enhances the relevancy and accuracy of generative responses. The way in which RAG system works is though combining two main elements: a retriever, which recognizes and retrieves appropriate documents or snippets in an external corpus, and a generator, which is usually an LLM, that would synthesize a reply based on both the initial prompt and the context extracted [2]. This architecture allows models to get access to real-time or domain-specific information, which is intermediating concept between parametric and non-parametric knowledge representations [2][3]. Although the classical form of RAG has been shown to be effective in enhancing factual consistency, it usually makes use of strict, linear retrieval-generation pipelines that are ineffective in tasks that require iterative reasoning or dynamic information [3][4].

In parallel with development of RAG, there are also agentic paradigms, in which LLMs are incorporated into autonomous agents with the ability to reason, plan, use tools and interact with the environment [5][6]. Such agents include frameworks like AutoGPT and LangChain, which are specifically aimed at decomposing complicated tasks into sub-tasks, communicate with other APIs or other tools, and adjust their behaviors on reacts or changing objectives [5][7]. When these agentic powers are formalized into the RAG system, they become Agentic RAG: an intergenerational system in which retrieval, generation and decision-making all happen through goal-oriented iterative process instead of fixed sequences [6][7].

The weaknesses of traditional RAG are also becoming more evident as the tasks that involve language are becoming more complex and ambitious [5]. The solution to these limitations is Agentic RAG, which adds autonomous self-directed behavior to the retrieval-generation loop to allow less rigid, context aware, and more robust AI systems [6][7]. The current review will set out to give a detailed discussion of the Agentic RAG paradigm, its architecture and building blocks, and practical applications. We aim to inform both scholars and practitioners in the overlapping of retrieval systems, language modelling and autonomous AI agents by mapping the situation at hand where it is likely to go.

2. Related Work

The history of Retrieval-Augmented Generation (RAG) has been documented well enough, with classic-style architectures that employed TF-IDF and BM25 to perform lexical retrieval [2], to dense-based retrieval, such as DPR [14][4][3], that pointed to the fundamental flaws of the static-RAG models, namely, the inability to support the iterative-refinement and multi-step-reasoning paradigms. To address these obstacles, modular RAG [16] came into existence, where component-based, adaptable architectures were used, and Graph RAG [17] was using knowledge graphs to aid relational understanding. Meanwhile, agentic paradigms began to emerge due to the emergence of agents such as AutoGPT and LangChain, which have the ability to plan, maintain memory, and use tools. It resulted in the creation of Agentic RAG [7], which integrates dynamic retrieval methods with autonomous decision-making to optimally guide complex and multi-step processes.

3. Background

Retrieval-augmented generation (RAG) is a paradigm of LLM augmentation by non-parametric sources of external knowledge, which enhances factual accuracy and prevents hallucinations. RAG allows models to support user queries that need recent data by adding a retrieval mechanism to the generation process, thereby making them more responsive to user queries [2].

3.1 Overview of RAG Architecture

The traditional RAG structured consists of three fundamental elements which work in pipeline sequence: retriever, a vector store and a generator.

- **Retriever:** When a user query is sent, the retriever module inspects an outside body of knowledge, usually a document corpus which has been indexed with the help of a vector representation of semantically important information. Initial systems were based on sparse methods, such as TF-IDF or BM25, however, current systems tend to use dense retrieval over embeddings produced by models like Sentence-BERT or the embedding APIs provided by OpenAI [14].

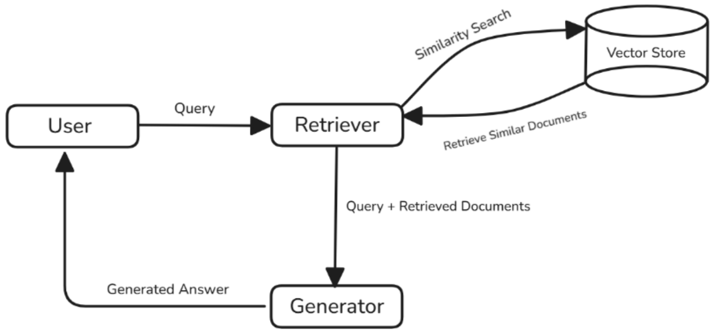

Fig. 1. Overview of Traditional RAG Architecture

- **Vector Store:** This is an element which forms the backbone of the retrieval system. Pre-computed document embeddings are stored in databases like FAISS, pinecone, Chroma, and Weaviate and can be used to quickly and effectively similarity search at query-time retrieval [2].

- **Generator**: The documents retrieved are joined with the initial query to create an augmented prompt that is inputted into a language model like GPT-4, LLaMA 3, or Claude 3. The model has a response based on the retrieved context thus generating response outputs which are more informed and based on current or domain specific knowledge [14].

This pipeline allows RAG systems to mitigate some of the shortcomings of closed-book language models by anchoring generation in verifiable external sources.

3.2 Limitations of Static Retrieval

Although the traditional RAG systems have their benefits, they have a number of structural limitations that limit their efficiency, especially in the context of processing complex, dynamic or ambiguous queries:

- **Lack of Iterative Refinement:** Retrieval is normally one-shoting in the retrieve-then-generate model. In case the original search is inadequate or a touch off-target to the query purpose, the system cannot narrow down or rectify its search decision in the middle of the search [4][7].

- **Inadequate for Complex Reasoning:** A lot of real questions need synthesis of a variety of sources or in a series of steps of reasoning. Such multi-hop reasoning is not well supported by the use of Static RAG pipelines which are unable to break down tasks of dynamically align information collection [4][6].

- **Fixed Retrieval Strategy:** Traditional RAG methods fail to adjust their retrieval logic depending on intermediate retrieval outcomes, which determines their performance on tasks whose solution is conditional exploration on multi-turn interaction [7].

- **Limited Tool Use:** Classical RAG systems lack any external tools web APIs, code interpreters or calculators, which limits their capability to process operations involving procedural operations or access to real-time data [4][6].

3.3 Evolution of RAG Paradigms

There has been a notable evolution in the field of Retrieval-Augmented Generation (RAG), due to the more demanding demands of real-world tasks where high levels of contextual fidelity, ability to scale to operational scale, and multi-step complexity of reasoning are required. Early RAG systems, which are founded on the simplistic retrieval mechanism of using keywords, have evolved into sophisticated and agile structures. Moderns systems are often designed in a modular fashion, and are integrated with heterogeneous sources of data, and can have autonomous decision-making capabilities, which makes clear the need to ensure that RAG models can skillfully handle complex query forms [7]. In this part, the evolutionary pathway is outlined, focusing on the most important paradigms, i.e. Naïve RAG, Advance RAG, Modular RAG, Graph RAG, and Agentic RAG, explaining the main feature of each of these paradigms, its merits, and its limitations. A review of this development throws light into the progress made in both the retrieval mechanisms as well as the generative processes.

3.3.1 Foundational RAG: The Naïve Implementation

The basic RAG architecture, commonly called the Naïve RAG [14], implemented the basic retrieve-then generated process. In such systems, the traditional algorithms of lexical matching, such as TF-IDF and BM25 are usually used to find related documents in static collections. The reconstructed textual artifacts then added to the input context of the generative phase of the language model.

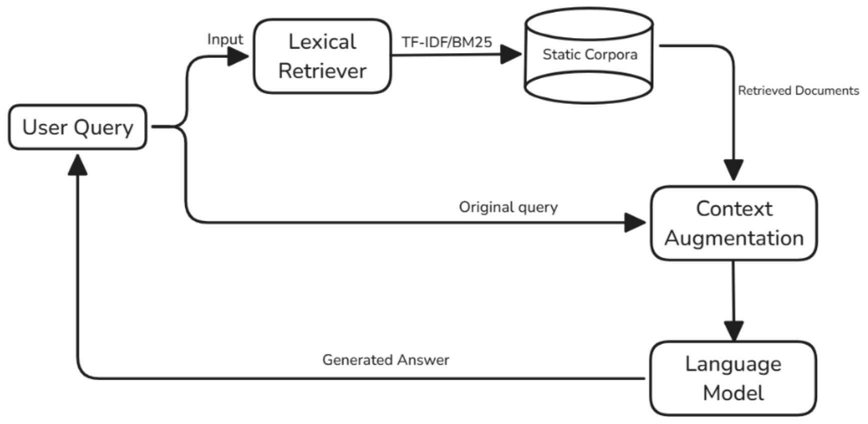

Fig.2. Overview of Naïve RAG Architecture

Though the implementation simplicity and viability of Naïve RAG were limited to elementary, face-based queries with low contextual complexity, Naïve RAG showed significant disadvantages:

- **Inadequate Contextual Apprehension:** The use of lexical overlap, instead of semantic congruence, often provided retrieved documents which had no faith to the subtle semantics of the user query.
- **Output Fragmentation:** Because of the lack of advanced preprocessing or contextual integration methodologies, it was frequently the case that generated responses were disjointed or were too generally applied.

- **Scalability Impediments:** Keyword-based methods of retrieval proved to be impractical in their performance and in the determination of relevance in the face of large-scale datasets.

Despite these shortcomings, Naïve RAG was an important step, which confirmed the possibility of merging retrieval with generation and provided the framework of ideas that underpin the further, more sophisticated paradigms.

3.3.2 Enhancing Semantic Fidelity: Advanced RAG

In an attempt to minimize the drawbacks associated with the Naïve RAG approach, Advance RAG [15] implemented strategies that focus on the increased semantic comprehension and accuracy of retrieval. Such systems shifted to using dense retrieval models, such as Dense Passage Retrieval (DPR) which projects queries and documents to high-dimensional vector representations. This help in similarity semantically, beyond the relationship of lexical similarity. The salient characteristics of Advance RAG include:

- **Dense Vector Representations:** This allows much better matching of intentionality of query and semantics of documents through vector space models.

- **Context-Sensitive Re-ranking:** It makes use of neural network models to re-rank the original set, and documents with a high degree of contextual relevance get prioritized.

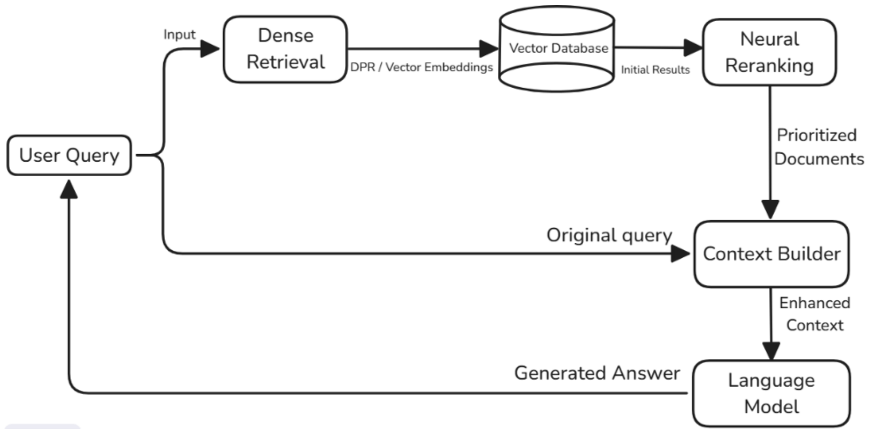

Fig. 3. Overview of Advanced RAG Architecture

These improvements make Advance RAG more effective when using it in applications where precision is required and subtle context interpretation, like automated research synthesis or a recommendation engine designed specifically to suit a particular user. Nevertheless, challenges related to computational time and effective scalability, especially when working with large datasets or with complex multi-step inferential queries, might still be relevant.

3.3.3 Architecture Flexibility s Task Adaptability: Modular RAG

Modular RAG [16] is based on a more recent paradigm that emphasizes the flexibilities and customizability of architecture. In this methodology, the pipeline is represented by individually manageable and potentially replaceable constituent parts of the RAG. The mentioned modularity allows optimization of components in specific domains or specific task requirements. Significant progress in the area of Modular RAG is:

- **Hybrid Retrieval Strategies:** More often that not, adopts a synergistic combination of both sparse (e.g., BM25 variants) and dense retrieval mechanisms [7] (e.g., DPR) to design retrieval fidelity to heterogenous types of queries.

- **External Tool and API Integration:** Has the ability to integrate external Application Programming Interfaces (APIs), databases, or custom computational modules, thus allowing retrieval fidelity to heterogeneous types of queries.

- **Composable Pipeline Architectures:** Allows individual modules (retrievers, generators, rankers) to be substituted, augmented or reconfigured, at low cost, to allow high adaptability to given deployment situations.

An example of a financial analytics-oriented Modular RAG system could be an example that ingests real-time market data via an API, performs historical trend analysis using dense retrieval and synthesizing actionable investment intelligence using a customized generative model. This design philosophy makes the Modular RAG best applicable in the multi-domain application with a complex need of both the granular subject being precise and the high scalability requirements.

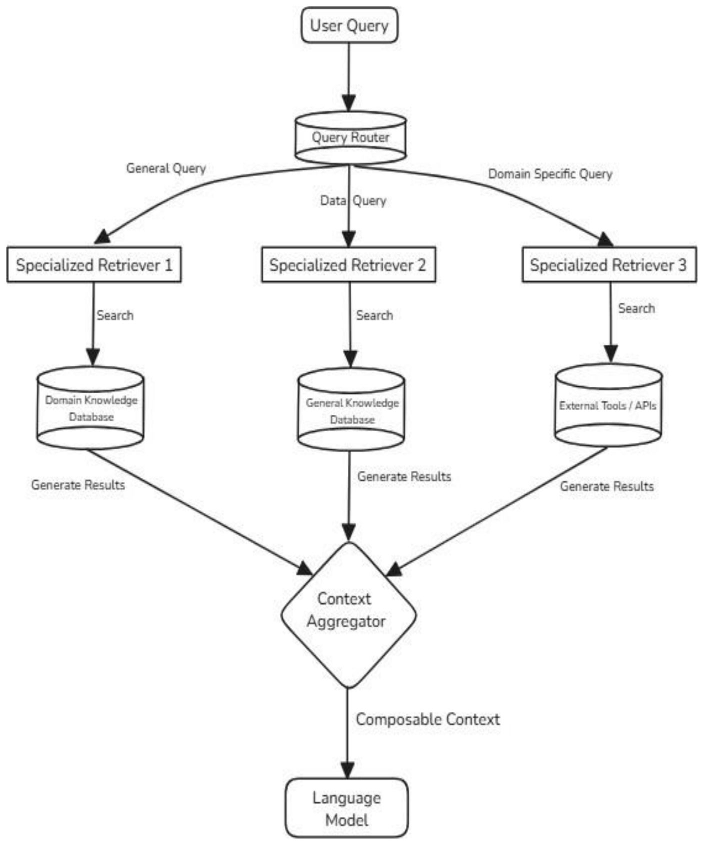

Fig. 4. Overview of Modular RAG

3.3.4 Leveraging Relational Structures: Graph RAG

Graph RAG [17] to extend conventional RAG with use of graph-based data structures, including knowledge graphs. Such systems can also increase the multi-hop reasoning and contextual enrichment of relational queries, especially those whose predicates are defined in relational terms. The distinguishing aspects of Graph RAG are:

- **Relational Inference Capabilities:** Ability to traverse and use connective topology (edges) between objects (nodes) in the graph structure.

- **Hierarchical Knowledge Representation:** Luminous ability in dealing with both structured and unstructured information that is stored through graph-based hierarchies.

- **Contextual Augmentation:** Strengthens the generative context with insights of relationships formed by traversal pathways of the graphs.

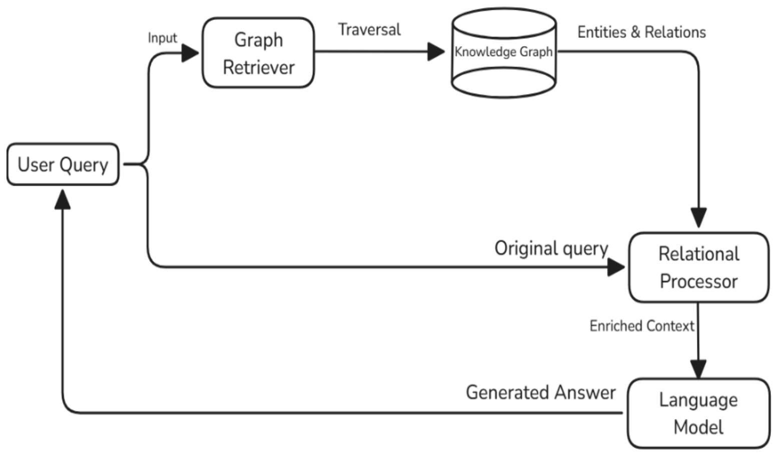

Fig. 5. Overview of Graph RAG Architecture

Graph RAG, on the other hand, is faced by limitations:

- **Scalability Problems:** The performance of the system can be limited because of size and the complexity of the underlying graph data structures.

- **Data Quality Dependency:** The effectiveness is very dependent on the quality and well-constructed graph representations which are of high fidelity.

- **Complexity of Integration:** Multi-graph-based retrieval system with traditional text retrieval introduces extra complexities to the system design and implementation.

Graph RAG shows a high level of use in the area where relational reasoning is the primary one, such as healthcare diagnostics, or legal research.

3.3.5 Towards Autonomous Operation: Agentic RAG

The paradigm shift is agentic RAG [7], which utilises autonomous agentic structures in the RAG model. These agents are capable of active decision-making routines, adaptive modulation over the retrieval strategies, and optimization of the operational processes based on query complexity and real-time environmental conditions, and can therefore overcome fixed, pre-programmed pipeline setups. The characteristics of the Agentic RAG paradigm include:

- **Autonomous Strategy Management:** Agents autonomously choose and change retrieval methodologies according to query-characteristics that are evaluated [7].
- **Iterative Refinement Loops:** This includes feedback loops that allow the system to improve accuracy of retrieval and relevance of response repeatedly [4][7].
- **Dynamic Workflow Orchestration:** The agents are dynamically configured to order and execute tasks (e.g., retrieval, processing, synthesis, generation) which is beneficial in applications where latency is a crucial factor [5][6][16].

Although it has high functionality, Agentic RAG has special implementation issues:

1. **Complexity of Co-ordination:** The complexity of co-ordinating between two or more autonomous agents requires more complex orchestration mechanisms [6][13].
2. **Computational Overhead:** Dynamic and possibly multi-agent system requirements may increase system resources demands [16].
3. **Scalability at Load:** It is essentially scalable, but very high query rates may exert high demand on computational resources because of the dynamism in the processing requirements [2][14].

The agentic RAG has significant application potential in application scenarios that require a high level of adaptability and context sensitivity in volatile operational environments, including complex customer interaction systems, real-time financial analysis systems and adaptive learning systems [5][13].

3.4 Persistent Challenges in RAG Frameworks

Despite the significant improvements in the capabilities of Large Language Models that the application of RAG methodologies brings due to the external knowledge integration, there are certain inherent limitations in its use, especially when it comes to less advanced applications. These limitations are capable of reducing the effectiveness in challenging real-world applications.

3.4.1 Contextual Coherence and Integration

The obstacles that RAG systems face in their smooth incorporation of retrieved information into the end product of the generative output are a common challenge. Although relevant factual information may be successfully retrieved, the inability to provide information that is contextually sensitive due to the static architecture of certain pipeline architectures, or due to a lack of contextual apprehension, may result in the delivery of fragmented, inconsistent, or otherwise inappropriate response to the particularities of the query, usually leading to generalized assertions.

Considering the above, a system may bring information about new therapeutic interventions but be incompetent to integrate these results into a logical, context-specific explanation that can be applied to specific patient profiles [17].

3.4.2 Deficiencies in Multi-Hop Inferential Reasoning

Many practical questions require the search and integration of information that is spread over a variety of documents or other data points. Conventional RAG systems can be characterized by a lack of sophisticated processes and mechanisms needed to adjust retrieval strategies based on interim feedback or user feedback. This weakness inhibits the development of holistic, all-inclusive answers to complicated and multi-faceted questions. An example that can be used is the task of assessing transferability of European renewable energy policies to developing countries; this would require combining policy prescriptions, socio-economic contextual issues and economic impact evaluation, which is a synthesis task that is difficult when dealing with simple RAG architectures [3][4].

3.4.3 Scalability and Latency Constraints

The computation cost related to the search and ranking of massive datasets becomes intensively costly as the number of potentially useful sources of external data multiplies. This may introduce high latency in response generation, which is harmful in applications that need near real-time interaction, e.g. synchronous customer support or time-sensitive financial analytics, where response latency can cause sub-optimal performance or strategic opportunity loss [14][17].

4 Agentic RAG

The need to have systems able to autonomously search extensive information tasks through the ever-growing capabilities of language models has led to the emergence of a new category of architecture. An important advancement of the traditional RAG systems, the Agentic Retrieval-Augmented Generation (also known as Agentic RAG) is an approach that places the process of retrieval and generation within a wider cognitive loop of an autonomous AI agent [8][9]. This part presents the principles underlying agentic systems, emphasizes that Agentic RAG is not designed in conventional ways and discusses the reasons why this agentic type of design is motivated to appear.

4.1 Definition of Agentic Systems in AI

The agentic systems are AI designs that are expected to act with a high-level of autonomy towards the goals set. Instead, these systems have the ability to perceive a situation, think about the goals, decide and act instead of following a predetermined chain of actions, and they may also engage with tools or environments external to themselves [10]. The agentic systems are usually characterized by core capabilities of:

- **Planning:** Dividing high level tasks into manageable structured subgoals.
- **Tool Use:** Communicating with API, calculators, web search engines or code interpreters on-demand.
- **Memory:** The ability to store information that is relevant in interactions, both in the short term context and long term knowledge.
- **Reasoning:** Making logical conclusions based on context, previously acquired knowledge and experience.
- **Self-Correction:** Tracking the achievements and redesigning plans depending on the feedback or performance errors detected [11][12].

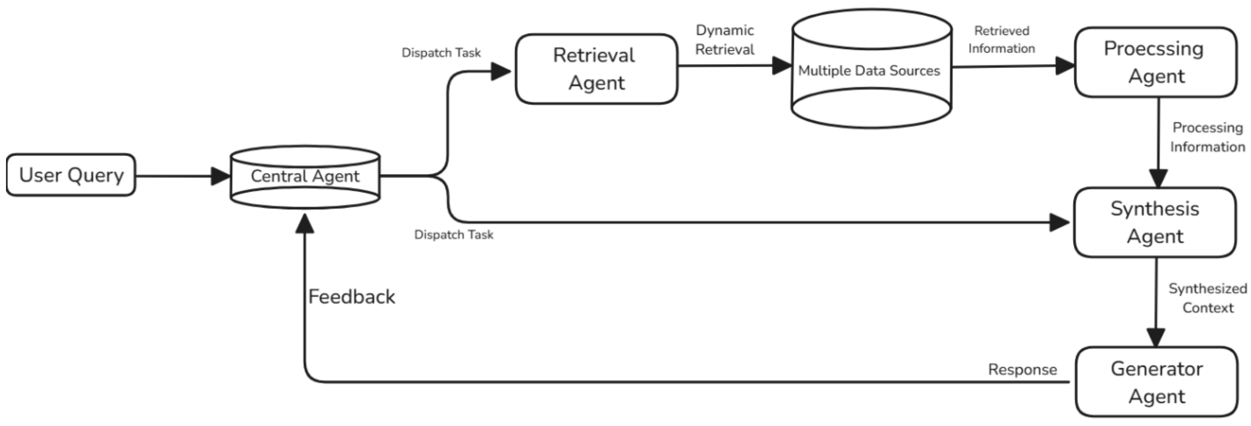

Fig. 6. Overview of Agentic RAG Architecture

Such abilities allow agentic systems to have goal direction behavior, changes in task needs, and act in dynamic or uncertain situations.

4.2 Key Differences from Traditional RAG

Unlike traditional RAG systems, which are based on a largely fixed, retrieve-then-generative model with very little adaptation and scarce decision making [2], Agentic RAG is based on an iterative and adaptive process that is governed by autonomous agents. This change presents a number of essential differences:

- **Dynamic Retrieval Strategy:** Agentic RAG systems have the capability of deciding what to retrieve, when to retrieve it, and how to narrow down search queries according to the developing task cognition [9].
- **Iterative Execution:** This is instead of a single step of retrieval but agents may execute multiple retrieval and reasoning steps modifying their strategy across the results along the way.
- **Multi-Tool Integration:** These systems do not consist of text-only retrieval. They have the choice and interface with a variety of tools (e.g., APIs, data parsers, calculators) to complete subtasks involving different modalities of processing [8].
- **Reasoning and Synthesis:** The data that is retrieved is not passively used. Agents are prolific in their reasoning activities that span across information sources, correlated facts, and synthesis of their findings to produce consistent and correct answers [12].

This loop of agents creates a degree of flexibility and strategic control, which are not present in the traditional RAG implementation.

4.3 Motivations Behind Agent-Based Enhancements

The development of Agentic RAG is highly driven by the increased complexity of real world information tasks which are difficult to answer using the static systems. Key drivers include:

- **Complex Query Processing:** Processes involving comparative analysis, research synthesis or even domain specific inquiry will be frequently met with the need to decompose goal, collect a wide variety of information and use the findings in multiple successive steps [13].
- **Enhancing Retrieval Precision:** Static RAG systems tend to access the irrelevant or non-sufficient context. The refinement loops can be improved using agentic RAG to enhance the relevancy and quality of the information utilized [11].
- **Improving Autonomy and Control:** Agentic RAG fosters a higher level of autonomy when dealing with different types of queries; to do this, the system is allowed to select its own strategy, be it to retrieve, use tools, or reason internally [10].
- **Expanding Capabilities Beyond Text:** Agents are able to do real-time calculations, access live data or call specialized services with access to external tools, greatly expanding the range of activities enabled by AI systems [9].

Finally, Agentic RAG will strive to develop more intelligent, adaptable and interactive systems that are more representative of the complexity of human-like decision-making, problem-solving in open-ended environments.

5 Agentic Workflow Architectures: Strategies for Dynamic LLM Application Structuring

5.1 Sequential Task Decomposition via Prompt Chaining

Prompt chaining is a workflow structure in which a complicated task is broken down into a sequence of sequential sub tasks [7]. The result is built up by the processing of the output of the previous step in the chain. This hierarchical and linear processing is what allows it to be more accurate because there is less complexity to address at each point. Nonetheless, the sequential dependency that may occur at the nature could add more end to end latency.

Applicability: The approach is especially effective when a key task can be divided in a logical manner into a set sequence of sub-tasks in which each of them adds up to the final product. It is beneficial in the situations when a stepwise reasoning is proven to enhance the output fidelity.

Illustrative Use Cases: Multi-stage content generation (e.g. writing marketing copy in one language and then translating it in an English variant). Structured document synthesis (e.g. first outline generation, then validation and then full-text development).

5.2 Input-contingent with Routing

The routing workflow pattern implies that first an input query or data is classified and then it is sent to a dedicated processing module, prompt, or downstream LLM customized to handle that type of input [7]. This is a strategy to make sure that the most suitable resources or logic paths handle heterogeneous inputs. The main advantages are efficiency in processing and quality of responses because tasks are handled in a specific manner.

Applicability: Routing architectures are best applied in applications where there are different input categories which require different handling protocols or computational resources. This guarantees the maximization of the performance features of various types of inputs.

Illustrative Use Cases: There are common uses where classifying customer service requests (e.g., technical support, billing, general information) is done, or dynamic allocation of computational resources by sending simple queries to smaller (and more affordable) models and larger (more capable) models expensive queries.

5.3 Enhancing Throughput via Parallel Processing

Parallelization as a strategy of workflow: It is a kind of subdivision of a task, where a sub-process is performed simultaneously into several others which are independent of each other [7]. This can be a highly effective method to reduce the overall system processing latency and boost system throughput. Techniques in this pattern are sectioning, in which independent subtasks are done in parallel, and voting (or ensemble methods), in which multiple models/processes are run concurrently on the same inputs to increase accuracy or confidence by voting/consensus or by voting/comparison.

Applicability: This trend is beneficial to application to tasks that can be subdivided into a separate computational unit and thus speeds up performance. It is also useful in cases where redundancy in terms of several parallel executions can enhance the output reliability or confidence scoring.

Illustrative Use Cases: Segregating is comparable to content moderation systems in which the screening of the input and the production of responses happen simultaneously. Voting mechanisms are used in such applications as security analysis where multiple models may be used to scan the code at the same time or robust content analysis where multiple moderation judgments are aggregated.

5.4 Dynamic Task Allocation: The Orchestrator-Worker Paradigm

This hierarchical workflow system consists of a central coordinating LLM known as the orchestrator that takes incoming tasks and dynamically breaks them down into subtasks and assigns these subtasks to special worker LLMs or modules [7]. The organizer then makes the synthesis of the outcomes of the workers. Unlike static parallelization, in this paradigm there is a flexibility to changes in input complexities as well as task structures.

Applicability: The orchestrator-worker model applies best in tasks that are complicated and whose decomposition is best done dynamically, and that need real-time adjustment, especially when the structure of subtasks is not pre-specified.

Illustrative Use Cases: The concepts have been used in complex software engineering, e.g. automatically changing interrelating files in a codebase in response to high-level change requests, and advanced research synthesis in real time, i.e. including dynamic information retrieval, filtering, and combination of information across multiple sources.

5.5 Iterative Refinement

The evaluator-optimizer process adopts a generation-refinement cycle [7]. The first output is obtained which is evaluated by a separate evaluator model or process, according to a predefined criterion or metric. The performance of this evaluation is used to perform a second optimizer step (typically by re-prompting the generator or re-prompting a different specialized model) to gradually improve the quality of the output in one or more cycles.

Applicability: The loop of this iterative refinement is especially useful when the quality of the output generated can be significantly enhanced by a series of successive improvements, and when objective or heuristic evaluation criteria can be necessary and reliably used.

Illustrative Use Cases: Use-cases include improving the quality of machine translation by repeating the process of testing against linguistic standards and then refining or performing a multi-step database querying or research problem whereby the results of one step are used to update the query in the next step.

6 Conclusion

The presented paper discussed the development of the traditional Retrieval-Augmented Generation to Agentic RAG, which is characterized by the autonomous planning, tool usage, and adaptive workflows. Introducing a solution to the rigidities of the previous RAG systems, Agentic RAG can provide a better performance on complex and dynamic tasks, including research automation and decision-making support. Even though it presents some additional difficulties in terms of the complexity of the system and ethical concerns, Agentic RAG can be viewed as one of the initial steps in the development of smarter and more reactive AI systems. Further development ought to be done in the future by extending capabilities to multimodal settings and self-directed agent loops.

7 References

[1] Aditi Singh. Exploring language models: A comprehensive survey and analysis. In 2023 International Conference on Research Methodologies in Knowledge Management, Artificial Intelligence and Telecommunication Engineering (RMKMATE), pages 1–4, 2023.

[2] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al., Retrieval-augmented generation for knowledge-intensive nlp tasks,

https://arxiv.org/pdf/2005.11401

[3] Weihang Su et al. (2025) – Parametric Retrieval Augmented Generation, https://arxiv.org/pdf/2501.15915

[4] Zhihong Shao[1], Yeyun Gong[2], yelong shen[3], Minlie Huang[1*], Nan Duan[2], Weizhu Chen[3]. Enhancing Retrieval- Augmented Large Language Models with Iterative Retrieval-Generation Synergy. https://arxiv.org/pdf/2305.15294

[5] Yu, T., Zhang, S., C Feng, Y. (2024). Auto-RAG: Autonomous Retrieval-Augmented Generation for Large Language Models. https://arxiv.org/abs/2411.19443.

[6] Wu, J., Zhu, J., C Liu, Y. (2025). Agentic Reasoning: Reasoning LLMs with Tools for the Deep Research. https://arxiv.org/abs/2502.04644.

[7] Singh, A., Ehtesham, A., Kumar, S., C Khoei, T. T. (2025). Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG. https://arxiv.org/abs/2501.09136.

[8] Shinn, N., Jernite, Y., C Bosselut, A. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. https://arxiv.org/abs/2303.11366.

[9] Yao, S., Zhao, J., Yu, D., Zhou, D., C Nie, J. Y. (2022). ReAct: Synergizing Reasoning and Acting in Language Models. https://arxiv.org/abs/2210.03629.

[10] Park, J., Kim, S., C Zettlemoyer, L. (2023). Generative Agents: Interactive Simulacra of Human Behavior. https://arxiv.org/abs/2304.03442.

[11] Schick, T., Dwivedi-Yu, J., C Schütze, H. (2023). Toolformer: Language Models Can Teach Themselves to Use Tools. https://arxiv.org/abs/2302.04761.

[12] Karia, N., Patel, J., C Kumar, R. (2024). Self-Correcting Language Agents: A Survey of Memory-Driven Systems. Proceedings of the AAAI Conference on Artificial Intelligence. https://arxiv.org/abs/2401.04659

[13] Gao, L., Zhang, W., C Liu, Y. (2023). Towards Advanced AI Planning: Autonomy in Retrieval-Augmented Systems. https://arxiv.org/abs/2303.12345.

[14] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrievalaugmented generation for large language models: A survey, 2024. https://arxiv.org/pdf/2312.10997

[15] Eibich, M., Nagpal, S., C Fred-Ojala, A. (2024). ARAGOG: Advanced RAG Output Grading. https://arxiv.org/pdf/2404.01037

[16] Gao, Y., Xiong, Y., Wang, M., C Wang, H. (2024). Modular RAG: Transforming RAG Systems into LEGO-like Reconfigurable Frameworks. https://arxiv.org/pdf/2407.21059

[17] Peng, B., Zhu, Y., Liu, Y., Bo, X., Shi, H., Hong, C., Zhang, Y., C Tang, S. (2024). Graph retrieval-augmented generation: A survey. arXiv. https://arxiv.org/pdf/2408.08921