Evaluating RAG approaches for Question Answering

Maristella Agosti[1], Nicola D'Errico[2], Paolo Fantozzi[3], Donatella Firmani[4,‡] and Luigi Laura[2]

[1]*Università di Padova*
[2]*Università Telematica Internazionale Uninettuno*
[3]*LUMSA University*
[4]*Sapienza Università di Roma*

Abstract

Retrieval-Augmented Generation (RAG) systems have emerged as a powerful solution to enhance Large Language Models (LLMs) by grounding their responses in external knowledge sources, thus mitigating hallucinations. However, selecting the most effective RAG architecture for specific Question Answering (QA) tasks remains a challenge due to the variety of available techniques. This paper presents a comparative study of six different RAG systems, ranging from basic vector search and large context models to advanced architectures such as Sentence Window Retrieval, Auto-Merging, Knowledge Graphs, and Agentic RAG. We designed and evaluated 15 system variants—selected from approximately 230 preliminary trials—on a specific corpus of scientific and technical documents, using the "RAG triad" metrics (Context Relevance, Groundedness, Answer Relevance) provided by the Trulens framework with an LLM-as-a-judge approach. Our experimental results indicate that Knowledge Graph-based systems (RAG 4) generally offer the best balance of performance metrics. However, Sentence Window Retrieval (RAG 2) proves highly effective for broad, conceptual questions. We also discuss the potential of Agentic RAG systems and the trade-offs between accuracy, cost, and latency.

Keywords

Retrieval Augmented Generation, Question Answering, RAG Approaches

1. Introduction

Since the seminal work of Lewis et al. [1], Retrieval-Augmented Generation (RAG) systems have emerged as the solution to provide "grounded" responses to user queries. These systems combine a Large Language Model (LLM) with a retrieval component, which selects the most relevant set of contents from previously indexed external knowledge sources (such as databases and private document corpora) to answer the user's query. A RAG system uses this content to build a reference context, which is inserted into a suitable prompt, through which the LLM model is encouraged to generate a response. The generation process is therefore defined as "augmented" (by the retrieval process), because the language model is induced to generate a response strictly constrained by the context provided as input, while being left free to express and synthesize it with the words it deems most appropriate, according to the natural language knowledge acquired during training and incorporated into its internal parameters.

This emerging property of LLMs to learn "on the fly" specific tasks, which has made the development of RAG systems possible, generally referred to as "in-context learning," is remarkable considering that the model parameters remain the same, while the tasks that an LLM can perform can change dynamically in the inference phase. When developing these systems, one of the challenges is determining the

IRCDL 2026: 22nd Conference on Information and Research Science Connecting to Digital and Library Science, February 19-20, 2026, Modena, Italy

[‡]The work of Maristella Agosti, Donatella Firmani, and Luigi Laura has been partially supported by "Knowledge Graph technologies for the next-generation Computing Continuum", Sapienza Research Project B83C2500088005. The work of Donatella Firmani was partially supported by the HORIZON Research and Innovation Action 101135576 INTEND "Intent-based data operation in the computing continuum", the Sapienza Research Projects "Trustworthy Technologies for Augmenting Knowledge Graphs" (B83C22007180001) and project APM "Automotive Predictive Maintenance" (Fondo per la Crescita Sostenibile - Accordi per l'innovazione).

agosti@dei.unipd.it (M. Agosti); nicderrico@gmail.com (N. D'Errico); p.fantozzi1@lumsa.it (P. Fantozzi); donatella.firmani@uniroma1.it (D. Firmani); luigi.laura@uninettunouniversity.net (L. Laura)
0000-0002-4030-0978 (M. Agosti); 0000-0002-5442-2239 (P. Fantozzi); 0000-0003-0358-3208 (D. Firmani); 0000-0001-6880-8477 (L. Laura)

effectiveness of the many solutions proposed by various RAG application development frameworks (including Langchain [2], LlamaIndex [3], and others). While individual variants may perform well in exemplary use cases, it is important to test various techniques in heterogeneous scenarios to ensure reliability.

The aim of this experimental work is to implement a series of basic variants of RAG systems, adopting technological solutions mentioned in the literature as particularly effective, and to compare them on a small corpus of documents (scientific articles and technical manuals) to facilitate rapid testing on datasets of questions at different levels of complexity. As an evaluation system, we chose the one provided by the Trulens library [4], which uses three basic metrics (Context Relevance, Groundedness, Answer Relevance) to capture the main failure modes of a RAG system. The approach of using an LLM as a "judge" to assign accuracy scores [5] is particularly suitable where "ground truth" references are not available.

The distinctive contribution of this work primarily concerns the testing and comparison of a series of RAG system technologies to prove their actual performance. Starting from approximately 230 preliminary trials exploring different hyper-parameter combinations, 15 system variants were selected for the final evaluation grid across three question categories, yielding 45 evaluated tests (see Appendix B for the complete results).

As observed by Domingos [6], induction is not a logical procedure that guarantees success; data alone are not enough to ensure a model will correctly answer a question never seen before. RAG systems can therefore be considered as automatic driving systems, processing a reliable map (the context) to allow the LLM to navigate towards the correct answer.

The article is structured as follows: Section 2 reviews the related work on RAG systems and their evaluation. Section 3 illustrates the experimental methodology and the design of the six RAG systems developed. Section 4 details the experimental results, whilst Section 5 presents the conclusions and future developments.

2. Related Work

Retrieval-Augmented Generation (RAG) is a powerful methodology that significantly enhances the capabilities of generative language models by integrating external information retrieval techniques [7, 8]. This approach addresses a critical challenge faced by standalone Large Language Models (LLMs): the tendency to produce responses that may not be grounded in facts, thereby reducing the incidence of factual inconsistencies and hallucinations [7, 9]. By coupling pretrained language models with non-parametric retrieval modules that fetch external evidence during inference, RAG systems offer greater transparency, factual grounding, and adaptability to evolving knowledge bases [7]. The core RAG system comprises two primary components: Retrieval and Generation [7]. The Retrieval component aims to extract relevant information from external knowledge sources, typically involving indexing and searching [10]. The Generation component utilizes the retrieved content and the user query to formulate coherent and contextually relevant responses [10].

The ability of LLMs to dynamically perform specific tasks during the inference phase, often referred to as "in-context learning," is essential for RAG systems, allowing them to augment the generation process while the model parameters remain static [11, 10]. RAG models leverage both parametric memory (knowledge stored in model weights) and non-parametric memory (external documents accessed via retrieval) [7]. This characteristic has enabled RAG to achieve notable performance improvements, particularly in the realm of domain adaptation and Question Answering (QA) tasks [9, 7].

2.1. RAG Architectures and Enhancements

Since the introduction of RAG, numerous architectural variants and enhancements have emerged, categorized broadly into retriever-centric, generator-centric, hybrid, and robustness-oriented designs [7]. Innovations frequently focus on optimizing the retrieval process, ranging from query enhancement to retrieval granularity optimization [7].

For systems requiring sophisticated handling of context, approaches like Sentence Window Retrieval (which retrieves small chunks and expands the context window with surrounding text blocks) and Auto-Merging Retrieval (which uses hierarchical indexing to merge related nodes for broader context) are employed to enhance context relevance and improve the signal-to-noise ratio [11, 7].

Furthermore, advanced retrieval methods leverage knowledge structures beyond flat passages. Knowledge Graph Retrieval systems construct graphs by extracting entities and relationships, combining vector search on text blocks with graph traversal to extract coherent multi-hop contexts [10, 11]. This structure-aware RAG allows reasoning over explicit relations, leading to potentially improved factual consistency [10].

An increasingly important architectural focus is the use of Agentic RAG, where a software agent orchestrates heterogeneous capabilities, including retrievers and tools, to form an adaptive execution plan [7]. This represents a move toward dynamic planning and multi-step reasoning, addressing complex retrieval needs and improving efficiency [7, 8].

2.2. Evaluation of RAG Systems

Evaluating RAG systems presents unique challenges due to their hybrid structure and dependence on dynamic knowledge sources [12, 13]. A robust evaluation framework must assess multiple interdependent factors, including retrieval relevance, the faithfulness of generated responses, and overall answer utility [7]. The core dimensions used for evaluation typically include Context Relevance (how pertinent the retrieved documents are to the query), Answer Faithfulness or Groundedness (whether the output is grounded in the retrieved evidence), and Answer Relevance (whether the output adequately addresses the query) [13, 7].

Frameworks like Trulens [13] decompose quality scores into these three key metrics, often referred to as the RAG Triad [13]. Automated evaluation frameworks commonly employ Large Language Models as "judges" to score outputs, particularly where obtaining comprehensive "ground truth" reference answers is difficult or costly [11, 10, 12]. However, implementing LLMs as evaluative judges introduces challenges in aligning automated assessment with human judgment and ensuring consistent evaluation across varied use cases [13].

The importance of evaluating RAG has increased in parallel with advancements in RAG methodologies, necessitating a comprehensive overview of evaluation challenges and benchmarks to advance the field [13, 12].

3. Methodology

The experimental work aimed at designing and carrying out a comparison of six RAG systems, measuring accuracy metrics (Context Relevance, Groundedness, Answer Relevance), Cost, and Latency.

Dataset. To evaluate the different RAG systems within the scope of the QA task, a specific evaluation dataset was built, consisting of a corpus of four PDF documents and three sets of test questions (A, B, C) of increasing complexity. The corpus composition is summarized in Table 1.

Ref.	Document	Type	Pages
1	A few useful things to know about ML [6]	Scientific article	10
2	AI: A Review of Progress and Prospects in Medicine and Healthcare [14]	Scientific article	23
3	Foundational Large Language Models and Text Generation [15]	Google white paper	86
4	Agents [16]	Google white paper	42

Table 1
Corpus of documents used for evaluation.

Evaluation protocol. For each of the three question categories, 20 questions were generated per document (5 per document), yielding 60 test questions per category and 180 total. Category A contains factual queries requiring extraction of precise information (names, dates, numbers). Category B contains general and conceptual questions requiring synthesis of multiple facts. Category C contains mixed queries combining aspects of A and B, including comparative questions across topics. Each system variant was evaluated against all 60 questions in each category, and results were aggregated and averaged by (system variant, question set). A subset of example questions is provided in Appendix A.

Terminology. Throughout this paper, a *system variant* denotes one specific RAG pipeline type (0–5) configured with a particular set of hyper-parameters (e.g., RAG 1 – V1: Chunk size = 1024, Top-k = 3). A *test* denotes one evaluation run of a system variant against one question category (A, B, or C). The 15 final system variants tested on 3 question categories yield 45 tests. On average, 5–6 preliminary trials per RAG type were conducted to explore hyper-parameter combinations before selecting the 15 variants for the final grid; these preliminary trials account for the approximately 230 total tests mentioned above.

System variants. The six systems compared are:

- RAG 0: Large Context, i.e. no RAG at all; that is why we labeled it as RAG 0.
- RAG 1: Basic Retrieval, i.e. the most basic RAG approach [1].
- RAG 2: Sentence window retrieval, i.e. retrieving small chunks and expanding the context window with surrounding text blocks [17].
- RAG 3: Auto-merging, i.e. using hierarchical indexing to merge related nodes for broader context [18].
- RAG 4: Knowledge graph, i.e. constructing graphs by extracting entities and relationships, combining vector search on text blocks with graph traversal to extract coherent multi-hop contexts [19, 20].
- RAG 5: Agentic RAG, i.e. a software agent orchestrating heterogeneous capabilities, including retrievers and tools, to form an adaptive execution plan [21, 22].

Common configuration and model choices. Since there are many variables involved, the different RAG systems were designed using an empirical approach based on a common "engine" consisting of a single pair (Embedding Model, LLM) for all cases: Embedding Model "text-embedding-3-small"[1] and LLM "gpt-4o-mini"[2] from OpenAI (except for RAG 0 where "gemini-2.5-flash"[3] from Google was also used for comparison). These models were selected because they represent cost-effective, widely accessible options suitable for systematic experimentation across many variants; however, different model choices could influence the absolute scores and the relative ranking of the systems. The LLM temperature was set to 0.1 to introduce a small amount of randomness for more natural responses. The key hyper-parameters explored include: Chunk size and Chunk overlap (RAG 1), Window size and Top-k/Top-n with Reranker (RAG 2), Hierarchical node sizes (RAG 3), Max triplets per chunk and Path depth (RAG 4), and Tool set composition (RAG 5).

A total of 15 system variants were selected for the final evaluation grid. Approximately 230 tests were carried out during the overall experimentation (including preliminary trials), resulting in 45 final evaluated tests (15 variants $\times$ 3 question categories), whose complete results are reported in Appendix B.

The development environment setup included:

- MacBook Pro with M4 Pro processor, 48 GB RAM.
- Jupyter Notebook Python 3.12.7.
- Llamaindex Framework.
- Evaluation Library Trulens.

[1] https://platform.openai.com/docs/models/text-embedding-3-small
[2] https://platform.openai.com/docs/models/gpt-4o-mini
[3] https://deepmind.google/models/gemini/flash/

3.1. RAG 0 – Large Context

This system represents a "level 0" RAG, where the Retriever retrieves documents from the corpus and passes them entirely to the LLM generator, which must have a sufficiently large context window (e.g., 2M tokens for Gemini). Indeed, this is not a true RAG system, but it is a baseline for comparison. While huge context windows allow processing large documents, RAG systems still offer advantages in cost, latency, and avoiding "lost in the middle" phenomena. Two models were used: "gemini-2.5-flash" (Google) and "gpt-4o-mini" (OpenAI), configured to pass the entire corpus as a single large chunk.

3.2. RAG 1 – Basic Retrieval

This system implements a basic RAG module [1] with a functional retrieval component using semantic vector search. The input text is divided into chunks, converted into embedding vectors, and stored in a vector index. The system seeks to improve the signal-to-noise ratio by selecting only relevant content. Three variants were developed adjusting chunk size and Top-k parameters.

3.3. RAG 2 – Sentence Window Retrieval

This system improves performance by enriching the context using a "sentence window retrieval" technique [17]. The input text is divided into small chunks for precise retrieval. During the query phase, the retrieved blocks are expanded by including a window of preceding and following text blocks. This technique is implemented using `SentenceWindowNodeParser` and `MetadataReplacementPostProcessor`. A Reranker is also used to reclassify relevance.

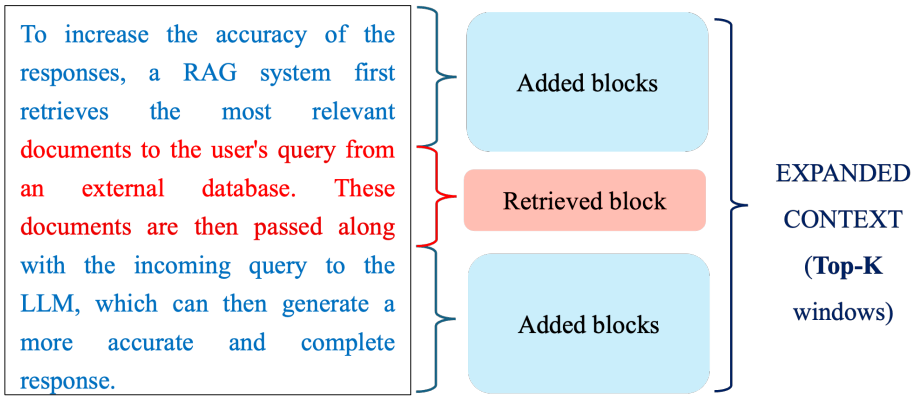

Figure 1: Relevant Node Retrieval and Context Expansion

3.4. RAG 3 – Auto-Merging Retrieval

The **Auto-Merging Retrieval** technique (also known as Hierarchical Retrieval) employs a multi-level indexing strategy designed to balance precise semantic search with the need for comprehensive context. In this architecture, documents are segmented into a tree structure of parent and child chunks—for example, larger parent blocks (e.g., 2048 or 512 tokens) are recursively divided into smaller leaf nodes (e.g., 128 tokens). While vector similarity search is performed on the granular leaf nodes to ensure high sensitivity to specific query details, a consolidation mechanism serves as a post-processing step: if a threshold of child nodes belonging to the same parent is retrieved, they are automatically "merged" and replaced by their original parent chunk [18]. This approach effectively reconstructs the broader narrative context from fragmented matches, providing the Large Language Model with a cohesive and continuous block of information that reduces the risk of losing critical surrounding details.

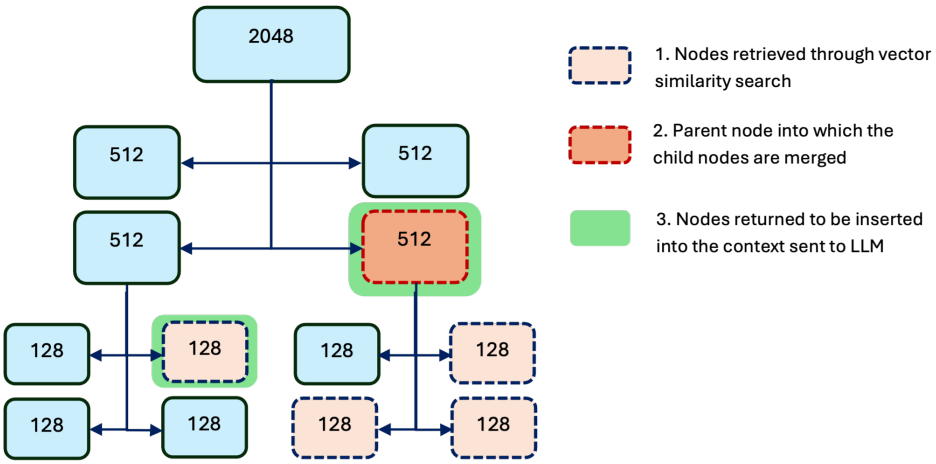

Figure 2: Auto-Merging Retrieval schema

3.5. RAG 4 – Knowledge Graph Retrieval

The **Knowledge Graph Retrieval** system leverages a structured representation of information to enhance retrieval accuracy and contextuality. During the indexing phase, an LLM extracts entities and their relationships (in the form of subject → predicate → object triplets) from the source text, constructing a rich knowledge graph. Retrieval is not limited to simple keyword or vector matching; instead, it combines vector search on text blocks with graph traversal techniques [19, 20]. This allows the system to retrieve not only the most semantically similar nodes but also their connected neighbors up to a configurable depth (e.g., retrieving related concepts linked by explicit relationships). By using a `PropertyGraphIndex` and performing a hybrid search (keywords + embeddings), this approach ensures that the retrieved context is both semantically relevant and structurally coherent, facilitating multi-hop reasoning.

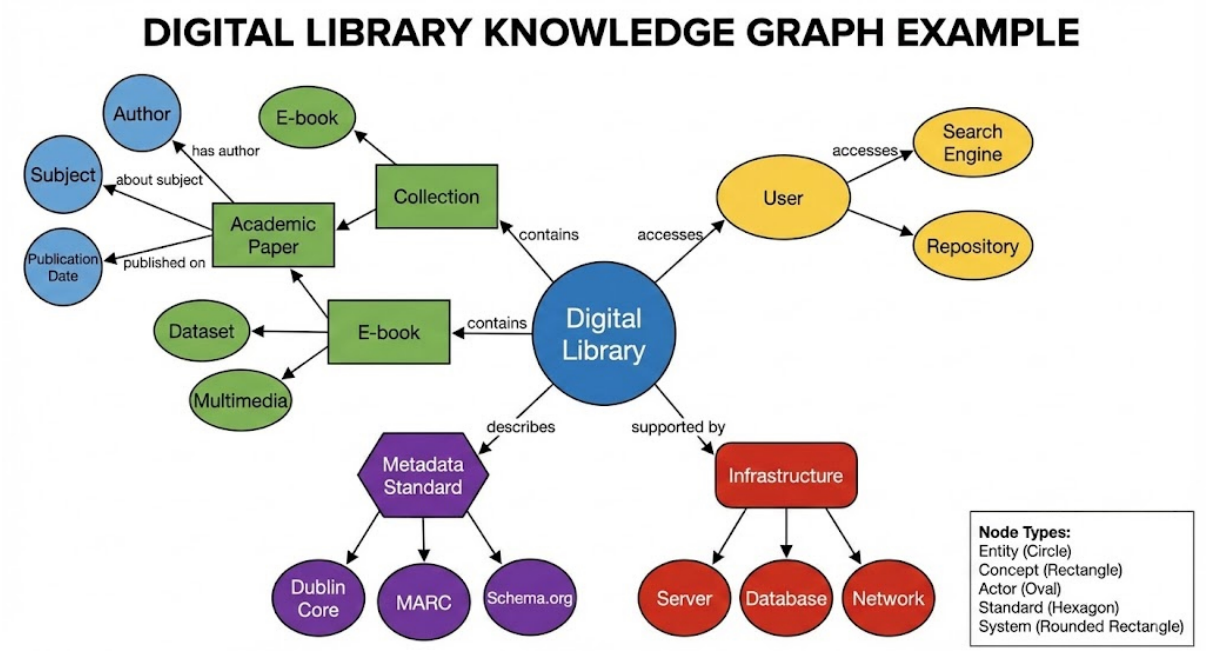

Figure 3: An example of a basic Knowledge Graph about Digital Libraries

3.6. RAG 5 – Agentic RAG

The **Agentic RAG** approach introduces a higher level of autonomy by employing a software "agent" capable of dynamic decision-making and tool orchestration. Unlike standard linear RAG pipelines, an agentic system utilizes a "reasoning loop" to analyze the user's query, determine the necessary steps, and iteratively execute specific tools—such as retrieving documents, performing calculations, or querying external APIs—to gather information [21, 22]. This capability allows the model to decompose complex problems and perform multi-step reasoning. In our implementation, we utilized the `OpenAIAgent` class with function calling capabilities, providing the agent with a suite of tools that wrap the other RAG subsystems (e.g., variants of RAG 2, 3, and 4), thereby enabling it to autonomously select the most appropriate retrieval strategy for each specific query.

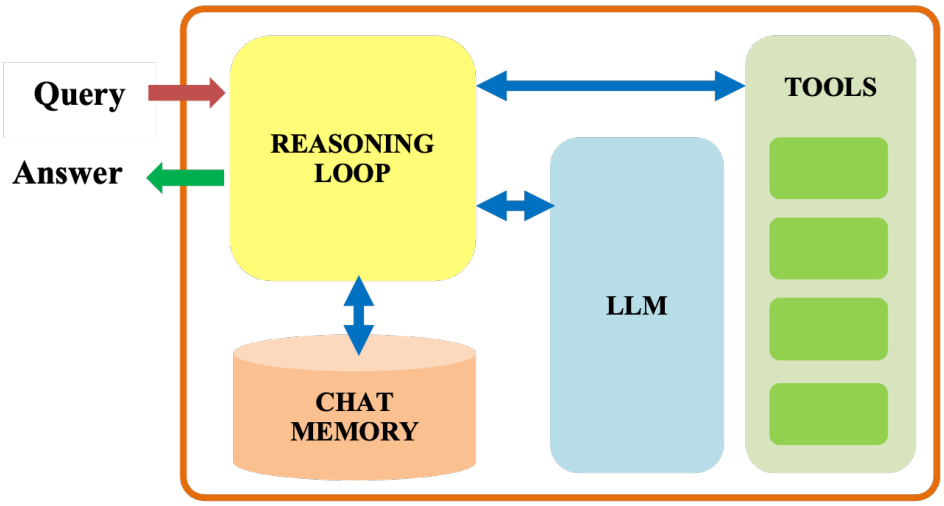

Figure 4: Basic structure of an agent

4. Results

The evaluation was conducted using the **Trulens** library, which employs the "RAG Triad" metrics to assess the quality of the system's output: Context Relevance, Groundedness, and Answer Relevance [4].

Context Relevance measures the pertinence of the retrieved information to the user's query, ensuring that the retrieved chunks contain data useful for answering the question without introducing noise. **Groundedness** (or Faithfulness) evaluates the extent to which the generated response is supported by the retrieved context, verifying that the LLM does not hallucinate information outside the provided source material. **Answer Relevance** assesses how effectively the generated response addresses the original user question, ensuring the answer is not only factually correct but also helpful and directly responsive to the user's intent.

In addition to these quality metrics, the system's efficiency was monitored by measuring **Cost** and **Latency**. Latency was measured as client-side wall-clock time on the machine where the RAG pipeline code runs; it therefore includes retriever calls, LLM generation, tool calls (if any), and judge calls, but does not account for variable network conditions. Cost was derived from token usage (prompt + completion tokens per LLM call) using the model pricing tables, including retrieval-augmented prompts, tool-augmented prompts, generation, and judge calls; no retrieval costs (in-memory vector database) or indexing costs were considered. Full cost and latency figures for all 15 system variants are reported in Appendix B.

In the following tables, the best result for each column within a given question set is highlighted in **bold**. In some cases, for single-row entries, the best result is the only one available, so all the values are bold. Tables 2–7 report representative results for each RAG system; the complete set of results for all 15

variants across all three question categories is provided in Appendix B.

4.1. RAG 0 Evaluation

Good Answer Relevance and Groundedness were observed, except for Category A questions where hallucinations occurred. Context Relevance was conventionally set to 0.5.

App Version	Set	Ans. Rel.	Ctx. Rel.	Ground.	Quality Score
V1: Gemini	A	0.967	**0.500**	**0.768**	**0.719**
V2: OpenAI	A	**1.000**	**0.500**	0.733	0.716
V1: Gemini	B	0.983	**0.500**	**0.974**	**0.782**
V2: OpenAI	B	**1.000**	**0.500**	0.918	0.771
V1: Gemini	C	0.983	**0.500**	**0.952**	**0.776**
V2: OpenAI	C	**1.000**	**0.500**	0.916	0.771

Table 2
RAG 0 Results. V1 uses Gemini (gemini-2.5-flash) and V2 uses OpenAI (gpt-4o-mini) as the LLM generator. Context Relevance is conventionally set to 0.5 since no retrieval filtering is performed. *Ans. Rel.* = Answer Relevance; *Ctx. Rel.* = Context Relevance; *Ground.* = Groundedness; *Quality Score* = geometric mean of the three metrics.

4.2. RAG 1 Evaluation

Performance depends significantly on chunk size. Best overall results were obtained with Chunk size 1024 and Top-k 3.

App Version	Set	Ans. Rel.	Ctx. Rel.	Ground.	Q. Score
V1: C=1024, K=3	A	**0.950**	**0.511**	0.938	**0.769**
V2: C=512, K=5	A	0.933	0.430	0.980	0.733
V3: C=256, K=9	A	**0.950**	0.320	**1.000**	0.673
V1: C=1024, K=3	B	**1.000**	**0.667**	**0.913**	**0.847**
V1: C=1024, K=3	C	**0.950**	**0.656**	**0.908**	**0.827**

Table 3
RAG 1 Results (best per question category). C = Chunk size (tokens), K = Top-k retrieved chunks. Three variants were tested; full results in Appendix B. *Q. Score* = Quality Score (geometric mean of Ans. Rel., Ctx. Rel., Ground.).

4.3. RAG 2 Evaluation

RAG 2 systems showed excellent balance and robust performance, especially on complex queries (Categories B and C).

App Version	Set	Ans. Rel.	Ctx. Rel.	Ground.	Q. Score
V1: Win=2, K=7, N=3	A	**0.833**	0.478	**0.879**	**0.705**
V2: Win=4, K=5, N=2	A	0.667	0.458	0.835	0.634
V3: Win=7, K=3, N=1	A	0.650	**0.633**	0.600	0.627
V2: Win=4, K=5, N=2	B	**1.000**	**0.750**	**0.874**	**0.869**
V2: Win=4, K=5, N=2	C	**0.967**	**0.700**	**0.847**	**0.831**

Table 4
RAG 2 Results (best per question category). Win = Window size (number of surrounding sentence blocks), K = Top-k retrieved nodes, N = Top-n nodes after Reranker. Three variants were tested; full results in Appendix B.

4.4. RAG 3 Evaluation

Like RAG 2, Auto-merging performed well on broader questions (B and C).

App Version	Set	Ans. Rel.	Ctx. Rel.	Ground.	Q. Score
V2: [512,256,128]	A	**0.933**	**0.324**	**0.986**	**0.668**
V2: [512,256,128]	B	**1.000**	**0.510**	**0.921**	**0.777**
V2: [512,256,128]	C	**1.000**	**0.457**	**0.915**	**0.748**

Table 5
RAG 3 Results (best per question category). [512,256,128] denotes the hierarchical chunk sizes (parent, intermediate, leaf) in tokens. Top-k = 12, Top-n = 6 (after Reranker). Two variants were tested; full results in Appendix B.

4.5. RAG 4 Evaluation

Knowledge Graph systems achieved excellent results across all metrics. The best version used `max_triplets_per_chunk=8` and `path_depth=2`.

App Version	Set	Ans. Rel.	Ctx. Rel.	Ground.	Q. Score
V1: Trip=8, D=2	A	**0.933**	**0.833**	**0.992**	**0.917**
V1: Trip=8, D=2	B	**1.000**	**0.711**	**0.742**	**0.808**
V1: Trip=8, D=2	C	**1.000**	**0.656**	**0.841**	**0.820**

Table 6
RAG 4 Results (best per question category). Trip = max triplets per chunk (number of subject–predicate–object triplets extracted by the LLM per text chunk), D = graph traversal path depth. Top-K = 10, Top-N = 3. Two variants were tested; full results in Appendix B.

4.6. RAG 5 Evaluation

Agent-based systems showed good performance, especially Variant V3 (Tool Set 3). Context Relevance was often medium-low, likely due to the agent retrieving intermediate partial results, but this did not negatively impact the final response quality.

App Version	Set	Ans. Rel.	Ctx. Rel.	Ground.	Q. Score
V3: Tool Set 3	A	**0.867**	**0.685**	**0.931**	**0.821**
V3: Tool Set 3	B	**1.000**	**0.722**	**0.801**	**0.833**
V3: Tool Set 3	C	**1.000**	**0.619**	**0.788**	**0.787**

Table 7
RAG 5 Results (best per question category). Tool Set 3 equips the agent with five tools: one general-purpose RAG 1 pipeline plus four document-specific RAG 4 (Knowledge Graph) pipelines. Three variants were tested (Tool Set 1: RAG 2+3+4 tools; Tool Set 2: four document-specific RAG 1 tools; Tool Set 3: RAG 1 + four RAG 4 tools); full results in Appendix B.

4.7. Comparison of Results

The comparative analysis of the six RAG architectures reveals distinct patterns in performance driven by the underlying retrieval mechanisms. The **Knowledge Graph Retrieval (RAG 4)** system emerged as the overall top performer, exhibiting a robust balance across all three "RAG Triad" metrics. Its graph-based structure proved particularly advantageous for **Category A** questions, which demanded retrieval of specific entities and disparate relationships; here, RAG 4 minimized hallucinations and maximized *Groundedness* by traversing explicit connections that vector-only search might miss.

Conversely, for queries requiring broader synthesis or conceptual understanding—represented by **Categories B and C**—the **Sentence Window Retrieval (RAG 2)** and **Auto-Merging (RAG 3)** approaches demonstrated superior efficacy. RAG 2, in particular, excelled by expanding the context around a retrieval hit, thereby preserving the narrative flow and preventing the fragmentation of

key information. This suggests that while semantic vector search finds the "needle," mechanisms like window expansion are essential for reconstructing the "haystack" required for comprehensive answers.

The **Agentic RAG (RAG 5)** displayed high potential, effectively leveraging its reasoning loop to handle diverse query types. However, this flexibility generally incurs higher latency and token costs compared to fixed pipelines. The **Basic RAG (RAG 1)** and **Large Context (RAG 0)** baselines, while functional, often struggled to match the precision of the more architecturally sophisticated methods, highlighting that simply scaling context or chunk size is less effective than structurally optimized retrieval for complex domain-specific tasks. It might be interesting to test these preliminary findings against atypical and not yet addressed RAG applications, such as task recommendations in online judges [23].

These trends are visually summarized in the accompanying figures. Figure 5 aggregates the Quality Score across all query categories, illustrating the consistent superiority of the Knowledge Graph approach (RAG 4) and the competitive performance of Sentence Window Retrieval (RAG 2). The chart clearly highlights that while advanced methods generally outperform the baselines, the margin varies significantly by query type. Complementing this, the radar chart in Figure 6 offers a multi-dimensional view of the trade-offs, mapping the systems against the three axes of the RAG Triad. Here, the "shape" of each system's performance becomes apparent: RAG 4 maintains a broad, balanced coverage, whereas baselines like RAG 1 often exhibit a skewed profile, potentially scoring high in relevance but falling short in groundedness.

5. Conclusions

In this work, we explored the landscape of Retrieval-Augmented Generation through a comparative analysis of six distinct architectures. We acknowledge that our experimental evaluation, though extensive, is far from exhaustive; the combinatorial vastness of RAG hyperparameters, retrieval algorithms, and generation models means that no single study can claim complete coverage of the field.

However, the primary intent of this research is not to provide a detailed map of every possible configuration, but rather to offer a reliable compass for developers and researchers. Just as a compass does not depict every feature of the terrain but reliably points towards the destination, our findings pinpoint the most promising directions for specific needs: Knowledge Graph approaches (RAG 4) offer the "true North" for precision-critical entity queries, ensuring high groundedness, whereas context-window techniques (RAG 2) serve as the optimal bearing for broader conceptual synthesis. By

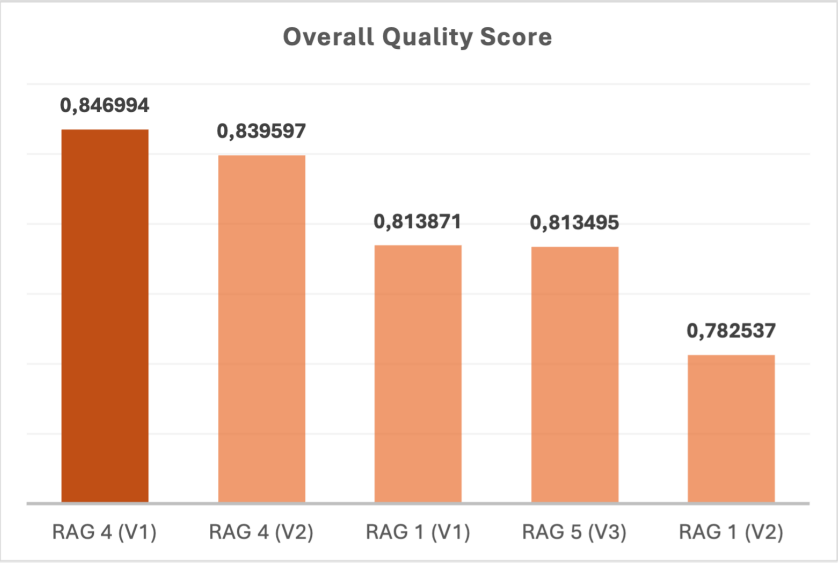

Figure 5: Overall Quality Score Comparison

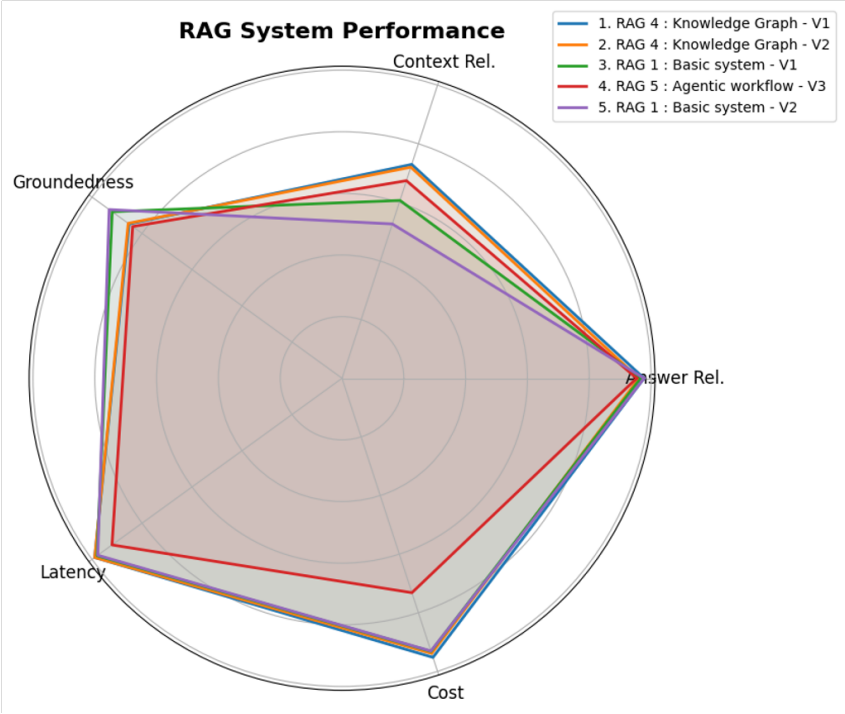

Figure 6: Radar Chart Comparison

aligning system design with these navigational aids, practitioners can better orient their efforts within the complex and rapidly traversing territory of advanced Question Answering systems, avoiding the pitfalls of generic implementations.

A. Example Evaluation Questions

Table 8 shows a subset of the evaluation questions generated for Document 1 of the corpus, illustrating the three categories of increasing complexity.

Cat.	Question
A	Who coined the expression 'curse of dimensionality'?
A	What is the title of the book authored by Vapnik published in 1995?
B	What is the fundamental goal of machine learning?
B	What is overfitting in machine learning?
C	What is the significance of Occam's razor in the context of machine learning and how does it relate to the trade-off between simplicity and accuracy in hypotheses?
C	What is overfitting in machine learning, and how can it be identified through the performance of a classifier on training and test data?

Table 8
Subset of evaluation questions for Document 1. Category A: factual queries; Category B: conceptual questions; Category C: multi-aspect and comparative questions.

B. Complete Evaluation Results

Table 9 reports the complete results for all 15 system variants evaluated on the three question categories (A, B, C), yielding 45 tests. Quality Score is the geometric mean of Answer Relevance, Context Relevance, and Groundedness.

System	Variant	Set	Ans. Rel.	Ctx. Rel.	Ground.	Q. Score
RAG 0	V1: Gemini	A	0.967	0.500	0.768	0.719
RAG 0	V2: OpenAI	A	1.000	0.500	0.733	0.716
RAG 0	V1: Gemini	B	0.983	0.500	0.974	0.782
RAG 0	V2: OpenAI	B	1.000	0.500	0.918	0.771
RAG 0	V1: Gemini	C	0.983	0.500	0.952	0.776
RAG 0	V2: OpenAI	C	1.000	0.500	0.916	0.771
RAG 1	V1: C=1024, K=3	A	0.950	0.511	0.938	0.769
RAG 1	V2: C=512, K=5	A	0.933	0.430	0.980	0.733
RAG 1	V3: C=256, K=9	A	0.950	0.320	1.000	0.673
RAG 1	V1: C=1024, K=3	B	1.000	0.667	0.913	0.847
RAG 1	V2: C=512, K=5	B	1.000	0.613	0.893	0.818
RAG 1	V3: C=256, K=9	B	1.000	0.527	0.951	0.794
RAG 1	V1: C=1024, K=3	C	0.950	0.656	0.908	0.827
RAG 1	V2: C=512, K=5	C	1.000	0.553	0.924	0.799
RAG 1	V3: C=256, K=9	C	1.000	0.498	0.923	0.772
RAG 2	V1: Win=2, K=7, N=3	A	0.833	0.478	0.879	0.705
RAG 2	V2: Win=4, K=5, N=2	A	0.667	0.458	0.835	0.634
RAG 2	V3: Win=7, K=3, N=1	A	0.650	0.633	0.600	0.627
RAG 2	V1: Win=2, K=7, N=3	B	1.000	0.661	0.823	0.816
RAG 2	V2: Win=4, K=5, N=2	B	1.000	0.750	0.874	0.869
RAG 2	V3: Win=7, K=3, N=1	B	1.000	0.817	0.773	0.858
RAG 2	V1: Win=2, K=7, N=3	C	0.950	0.606	0.796	0.771
RAG 2	V2: Win=4, K=5, N=2	C	0.967	0.700	0.847	0.831
RAG 2	V3: Win=7, K=3, N=1	C	1.000	0.783	0.793	0.853
RAG 3	V1: [2048,512,128]	A	0.917	0.347	0.919	0.664
RAG 3	V2: [512,256,128]	A	0.933	0.324	0.986	0.668
RAG 3	V1: [2048,512,128]	B	1.000	0.483	0.939	0.769
RAG 3	V2: [512,256,128]	B	1.000	0.510	0.921	0.777
RAG 3	V1: [2048,512,128]	C	1.000	0.444	0.897	0.736
RAG 3	V2: [512,256,128]	C	1.000	0.457	0.915	0.748
RAG 4	V1: Trip=8, K=10, N=3, D=2	A	0.933	0.833	0.992	0.917
RAG 4	V2: Trip=12, K=12, N=3, D=3	A	0.933	0.722	0.950	0.862
RAG 4	V1: Trip=8, K=10, N=3, D=2	B	1.000	0.711	0.742	0.808
RAG 4	V2: Trip=12, K=12, N=3, D=3	B	1.000	0.770	0.832	0.862
RAG 4	V1: Trip=8, K=10, N=3, D=2	C	1.000	0.656	0.841	0.820
RAG 4	V2: Trip=12, K=12, N=3, D=3	C	0.950	0.672	0.792	0.797
RAG 5	V1: ToolSet 1	A	0.950	0.623	0.846	0.794
RAG 5	V2: ToolSet 2	A	0.850	0.548	0.863	0.738
RAG 5	V3: ToolSet 3	A	0.867	0.685	0.931	0.821
RAG 5	V1: ToolSet 1	B	1.000	0.663	0.740	0.789
RAG 5	V2: ToolSet 2	B	1.000	0.706	0.770	0.816
RAG 5	V3: ToolSet 3	B	1.000	0.722	0.801	0.833
RAG 5	V1: ToolSet 1	C	1.000	0.617	0.722	0.764
RAG 5	V2: ToolSet 2	C	1.000	0.578	0.758	0.759
RAG 5	V3: ToolSet 3	C	1.000	0.619	0.788	0.787

Table 9
Complete evaluation results for all 15 system variants across the three question categories. RAG 5 Tool Sets: ToolSet 1 = {RAG 2, RAG 3, RAG 4}; ToolSet 2 = {4×RAG 1, one per document}; ToolSet 3 = {RAG 1 + 4×RAG 4, one per document}.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT in order to: Grammar and spelling check, Paraphrase and reword. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, D. Kiela, Retrieval-augmented generation for knowledge-intensive NLP tasks, in: Proceedings of the 34th International Conference on Neural Information Processing Systems, Curran Associates Inc., 2020. URL: https://proceedings.neurips.cc/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

[2] V. Mavroudis, LangChain, Technical Report, Alan Turing Institute, 2024. doi:10.20944/preprints202411.0566.v1, technical report. Available at Preprints.org.
[3] F. Gherghiou, Building Data-driven Applications with LlamaIndex, Packt Publishing, 2024. URL: https://www.packtpub.com/en-us/product/building-data-driven-applications-with-llamaindex-9781805124405.
[4] Trulens Project, Trulens: Evaluation framework for retrieval-augmented generation, https://www.trulens.org/, 2023.
[5] A. Datta, Benchmarking large language models as a judge: RAG triad metrics, Snowflake engineering blog, https://www.snowflake.com/en/engineering-blog/benchmarking-LLM-as-a-judge-RAG-triad-metrics/, 2025.
[6] P. Domingos, A few useful things to know about machine learning, Communications of the ACM 55 (2012) 78–87. doi:10.1145/2347736.2347755.
[7] C. Sharma, Retrieval-augmented generation: A comprehensive survey of architectures, enhancements, and robustness frontiers, arXiv preprint arXiv:2506.00054 (2025). URL: https://arxiv.org/abs/2506.00054.
[8] A. J. Oche, A. G. Folashade, T. Ghosal, A. Biswas, A systematic review of key retrieval-augmented generation (rag) systems: Progress, gaps, and future directions, arXiv preprint arXiv:2507.18910 (2025). URL: https://arxiv.org/abs/2507.18910.
[9] S. Rakin, M. A. Shibly, Z. M. Hossain, Z. Khan, M. M. Akbar, Leveraging the domain adaptation of retrieval augmented generation models for question answering and reducing hallucination, arXiv preprint arXiv:2410.17783 (2024). URL: https://arxiv.org/abs/2410.17783.
[10] A. Brown, M. Roman, B. Devereux, A systematic literature review of retrieval-augmented generation: Techniques, metrics, and challenges, arXiv preprint arXiv:2508.06401 (2025). URL: https://arxiv.org/abs/2508.06401.
[11] S. Li, L. Stenzel, C. Eickhoff, S. A. Bahrainian, Enhancing retrieval-augmented generation: a study of best practices, arXiv preprint arXiv:2501.07391 (2025). URL: https://arxiv.org/abs/2501.07391.
[12] A. Gan, H. Yu, K. Zhang, Q. Liu, W. Yan, Z. Huang, S. Tong, G. Hu, Retrieval augmented generation evaluation in the era of large language models: A comprehensive survey, arXiv preprint arXiv:2504.14891 (2025). doi:10.48550/arXiv.2504.14891.
[13] H. Yu, A. Gan, K. Zhang, S. Tong, Q. Liu, Z. Liu, Evaluation of retrieval-augmented generation: A survey, in: CCF Conference on Big Data, 2024. doi:10.1007/978-981-96-1024-2_8.
[14] S. Mishra, Artificial intelligence: A review of progress and prospects in medicine and healthcare, Journal of Electronics, Electromedical Engineering, and Medical Informatics (JEEEMI) (2022). doi:10.35882/jeeemi.v4i1.1.
[15] M. Barektain, et al., Foundational Large Language Models and Text Generation, White paper, Google, 2025. URL: https://www.kaggle.com/whitepaper-foundational-llm-and-text-generation.
[16] J. Wiesinger, P. Marlow, V. Vuskovic, Agents, White paper, Google, 2025. URL: https://www.kaggle.com/whitepaper-agents.
[17] P. Omrani, A. Hosseini, K. Hooshanfar, Z. Ebrahimian, R. Toosi, M. A. Akhaee, Hybrid retrieval-augmented generation approach for LLMs query response enhancement, in: 2024 10th International Conference on Web Research (ICWR), 2024. doi:10.1109/ICWR61162.2024.10533345.
[18] V. Krishnamurthy, V. Balaji, Yours truly: A credibility framework for effortless LLM-powered fact checking, IEEE Access (2024). doi:10.1109/ACCESS.2024.3520187.
[19] M. Li, S. Miao, P. Li, Simple is effective: The roles of graphs and large language models in knowledge-graph-based retrieval-augmented generation, arXiv preprint arXiv:2410.20724 (2024). doi:10.48550/arXiv.2410.20724.
[20] T. T. Procko, O. Ochoa, Graph retrieval-augmented generation for large language models: A survey, in: 2024 Conference on AI, Science, Engineering, and Technology (AIxSET), 2024. doi:10.1109/AIxSET62544.2024.00030.
[21] T. Yu, S. Zhang, Y. Feng, Auto-rag: Autonomous retrieval-augmented generation for large language models, arXiv preprint arXiv:2411.19443 (2024). doi:10.48550/arXiv.2411.19443.

[22] T. Nguyen, P. Chin, Y.-W. Tai, MA-RAG: Multi-agent retrieval-augmented generation via collaborative chain-of-thought reasoning, arXiv preprint arXiv:2505.20096 (2025). doi:10.48550/arXiv.2505.20096.

[23] P. Fantozzi, L. Laura, Recommending tasks in online judges using autoencoder neural networks, Olympiads in Informatics 14 (2020) 61–76. URL: https://ioinformatics.org/journal/v14_2020_61_76.pdf. doi:10.15388/ioi.2020.05.