Is GraphRAG Needed?
From Basic RAG to Graph-/Agentic Solutions with Context Optimization

Long Chen[1]*, **Ryan Razkenari**[1], **Yuxuan Zhou**[1], **Yuan Tian**[1],
Rahul Ghosh[1], **Venkatesh Pappakrishnan**[2], **Disha Ahuja**[2], **Vidya Sagar Ravipati**[1]*
[1]Generative AI Innovation Center, Amazon Web Services (AWS)
[2]Cisco Systems, Inc.
{longchn, razken, yuxuzh, ytianaws, rahulgh, ravividy}@amazon.com
{vpappakr, disahuja}@cisco.com

Abstract

As advanced RAG variants like GraphRAG and Agentic RAG emerge, one leading question is when and how to use them. Here, we introduce a framework for different RAG scenarios evaluation and comparison on semi-structured knowledge bases, including regular RAG, GraphRAG, Modular RAG and Agentic RAG. We provide implementation for 9 standardized RAG scenarios, and conduct experiments for a comprehensive comparison. These scenarios are designed for real use cases regarding data and domain restrictions, spanning from simple document-based retrieval to advanced features such as hybrid text-graph retrieval, integration with computed or pre-defined domain knowledge graphs, agentic multi-step planning, and agent-graph integration. Besides, we present a novel context engineering method for GraphRAG and Agentic RAG, addressing the context/memory overflow issues, efficiently managing text and graph retrievals with new representations and agentic loop design, leading to 19%-53% reduction on token usage. Moreover, further analysis identifies a retrieval-generation gap where expanded retrieval does not proportionally improve generation quality, suggesting retrieval-oriented metrics overstate advanced retrieval benefits. This work provides data-driven insights on when and how to use them for building production-ready intelligent RAG systems.

1 Introduction

Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) has become the dominant paradigm for grounding Large Language Model (LLM) outputs in external knowledge. Traditional RAG approaches have proven highly effective for question-answering tasks over unstructured textual corpora. The increasing complexity of real-world applications has highlighted the limitations of purely text-based knowledge representation, driving demand for more sophisticated approaches to handle semi-structured knowledge bases that contain both unstructured textual information and explicit relational data among entities. For instance, precision medicine queries such as "Which gene is involved in vesicle transport, located in the kinetochore, and participates in the antigen processing pathway?" require simultaneous reasoning over entity properties and multi-hop relationships that basic RAG systems struggle to provide.

These needs expose limitations of basic RAG when confronted with semi-structured knowledge bases. Recognizing these limitations, advanced RAG variants have emerged. GraphRAG (Peng et al., 2024) extends traditional retrieval by incorporating graph-based representations and reasoning capabilities, enabling navigation of entity relationships and multi-hop inference over knowledge graphs (KG). Modular RAG (Gao et al., 2024) introduces architectural flexibility by decomposing the pipeline into specialized, interchangeable components optimized for specific data types and query patterns. Agentic RAG systems (Singh et al., 2025) leverage autonomous agents capable of dynamic planning, tool utilization, and iterative reasoning, enabling sophisticated multi-step problem-solving that adaptively interacts with various knowledge sources and employs self-correction mechanisms.

However, the rapid adoption of these advanced methodologies raises a fundamental question: do these sophisticated architectures translate to meaningful improvements, and do standard evaluation metrics capture the differences? While these sophisticated systems promise enhanced reasoning capabilities over interconnected data, the practical benefits of these approaches over well-optimized baselines remain largely unexplored.

To address this gap, we introduce a comprehensive framework for evaluating, and comparing different RAG paradigms on semi-structured knowledge bases, encompassing traditional RAG,

*Corresponding authors.

Proceedings of the Fifth Workshop on Generation, Evaluation and Metrics (GEM), pages 428–442

GraphRAG, Modular RAG, and Agentic RAG approaches. Our work provides systematic implementation of 9 standardized RAG scenarios designed for real-world use cases with realistic data and domain restrictions, spanning from simple document-based retrieval to advanced capabilities including text-to-graph query translation, hybrid text-graph retrieval, integration with computed or pre-defined domain knowledge graphs, agentic multi-step planning, and sophisticated agent-graph integration patterns. Through experimentation on semi-structured knowledge base in the precision medicine domain, we deliver data-driven insights that illuminate when and how to strategically deploy these RAG variants for building production-ready intelligent systems. Besides, we propose a simple but novel context engineering method for GraphRAG and Agentic RAG, addressing the context/memory overflow issues, by leveraging more concise text and graph context representation, as well as a new agentic loop design pattern beyond ReAct (Yao et al., 2023). We also identify a retrieval-generation gap: end-to-end evaluation of LLM-selected entities during answer generation, rather than raw retrieval rankings, reveals that expanded retrieval does not translate to proportional gains in answer quality. This suggests that retrieval-oriented metrics overstate the benefit of advanced retrieval strategies. Our contributions provide practitioners with empirical guidance for making informed architectural decisions based on specific use case requirements, data characteristics, and performance constraints.

2 Related works

Semi-structured knowledge bases integrating text and relational data are increasingly important for knowledge-intensive applications. STaRK (Wu et al., 2024) provides a comprehensive benchmark for evaluating retrieval over such hybrid resources, while datasets like HotpotQA (Yang et al., 2018) and ComplexWebQuestions (Talmor and Berant, 2018) test multi-hop reasoning.

GraphRAG approaches leverage knowledge graphs for relationship-aware retrieval (Edge et al., 2024; Peng et al., 2024), while Agentic RAG introduces autonomous agents that dynamically orchestrate retrieval and reasoning (Singh et al., 2025). The intersection of these approaches with semi-structured knowledge bases presents unique opportunities to leverage both the structural precision of graphs and the flexibility of agent-driven retrieval, though the benefits and trade-offs of these methods compared to simpler baselines remain an open question that our work aims to address.

3 Methods

3.1 Problem definition

In this work, we focus on knowledge retrieval and question-answering on semi-structured knowledge bases with both textual and relational information. Following Wu et al. (Wu et al., 2024), considering a knowledge base which contains a collection of textual documents $\mathcal{D}$ and a knowledge graph $\mathcal{G} = (\mathcal{V}, \mathcal{R})$, where $d_i \in D$ is a textual document describing an entity i, and $v_i \in V$ is the corresponding entity node on KG, with $\mathcal{R}$ being a set of relations between different nodes. For example, in STaRK-Prime dataset (Wu et al., 2024), each entity (e.g., a gene/protein or disease) i has a textual document d_i that contains the corresponding entity description, and the corresponding node v_i on the knowledge graph G encodes its relations with other nodes such as "side effect", "parent-child", "phenotype present", etc. Given a user query q, which could contain certain entity and relation information and requirements, the goal is to retrieve the right set of entities and relations from the knowledge base to answer the user query.

Thus, this is a typical RAG problem which could be solved by different scenarios. However, the selection of RAG scenarios highly depends on both the availability of the textual and relational resources, as well as the complexity of the user query. In the following sub-sections, we will explain the 9 scenarios under three categories: Regular RAG, GraphRAG, and Modular & Agentic RAG.

3.2 Regular RAG scenarios

Scenario 1 - RAG with entity description documents only: This baseline scenario indexes only entity description documents. As shown in Figure 1, given a query, text documents are retrieved via vector similarity search and concatenated as context for LLM answer generation. For our task, the LLM is instructed to provide both the answer and the entity ids. This approach represents the most commonly used RAG approach, especially for the case without accessibility to pre-defined relational data. However, it faces limitations such as lack of relational context to handle complex queries.

Scenario 2 - RAG with documents containing both entity description and relations: This

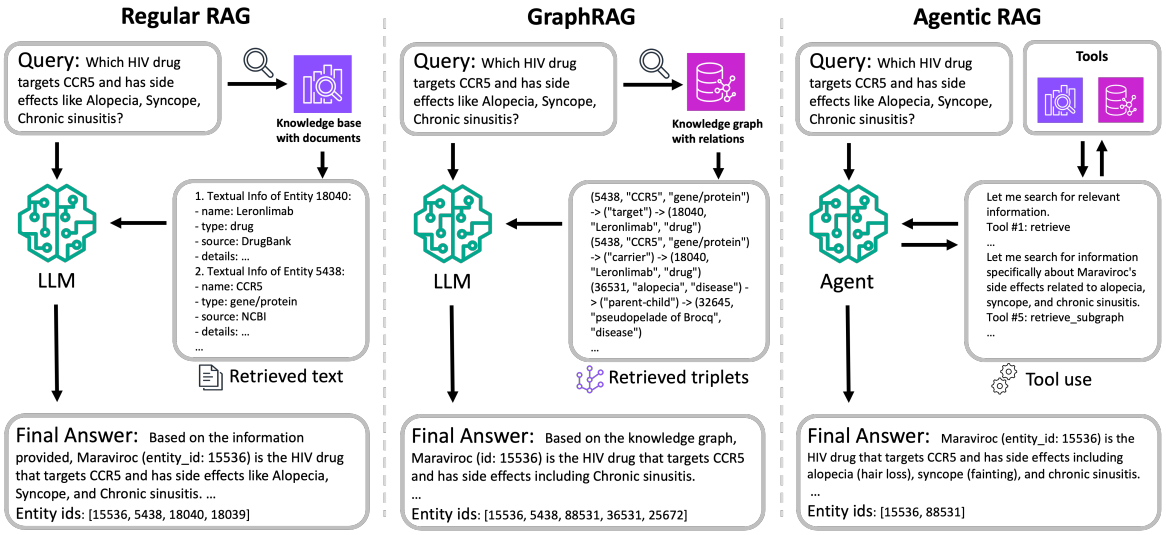

Figure 1: Comparison between Regular RAG, GraphRAG and Agentic RAG.

scenario augments each entity document with its 1-hop neighbors grouped by relation type before indexing. Figure 2 in Appendix provides examples of the documents in Scenario 1 and 2. The rest of the steps are the same as those in Scenario 1. This provides a simple way to incorporate relational information without graph search, though it is limited to 1-hop relations.

3.3 GraphRAG scenarios

Scenario 3 - GraphRAG with predefined knowledge graph only: In this baseline GraphRAG scenario, we utilize solely the predefined knowledge graph without incorporating any textual entity descriptions. The knowledge graph $G = (V, E, R)$ consists of vertices V representing entities, edges E representing relationships, and relation types R. The retrieval process operates directly on the graph through the following steps: (1) **Entity Extraction**: Given a user query q, we employ a LLM to identify and extract relevant entity names $E_q = \{e_1, e_2, \ldots, e_m\}$ mentioned in the query. (2) **Subgraph Extraction**: For each identified entity name $e_i \in E_q$, we perform k-hop traversal from the corresponding graph vertex with a maximum of N paths to extract relevant subgraphs $G^{(i)}_{sub} \subseteq G$. (3) **Graph Serialization**: The extracted subgraphs are converted into textual representations using triplet format: $\langle subject, predicate, object \rangle$, creating a linearized knowledge context C_{graph}. (4) **Answer Generation**: The serialized subgraph knowledge C_{graph} serves as context for the LLM to generate the final answer as well as the corresponding entity IDs. This scenario establishes the baseline performance when relying exclusively on structured knowledge, without using textual descriptions.

Scenario 4 - GraphRAG with computed knowledge graph from entity documents: Here, we construct a knowledge graph from entity description documents using automated knowledge extraction techniques, combined with vector-based retrieval from the document collection. The methodology consists of the following components: (1) **Entity Identification**: Apply named entity recognition (NER) techniques to identify entities E_{doc} from the document collection $D = \{d_1, d_2, \ldots, d_n\}$. (2) **Relation Extraction**: Utilize relation extraction models to identify semantic relationships between co-occurring entities within document contexts, extracting relationship triples $T_{extracted}$. (3) **Graph Construction**: Build the computed knowledge graph $G_{comp} = (V_{comp}, E_{comp}, R_{comp})$ where vertices and edges are derived from the extracted entities and relationships. (4) **Graph Refinement**: Apply entity resolution and relation consolidation techniques to merge duplicate entities and standardize relationship types. (5) **Hybrid Retrieval and Generation**: Combine retrieval results from both the vector index space and the computed knowledge graph for answer generation. This approach evaluates the efficacy of automatically constructed knowledge graphs combined with document-based retrieval.

Scenario 5 - GraphRAG with hybrid knowledge integration: This scenario combines both vector-based document retrieval and predefined knowledge graph traversal. The integration methodology involves: (1) **Vector-based Entity Retrieval**: Given a user query q, perform vector similarity search in the document index space to retrieve can-

didate entities and their corresponding textual documents $D_{retrieved}$. (2) **Subgraph Extraction**: For each identified entity name from the retrieved documents, perform h-hop traversal with a maximum of N paths in the predefined knowledge graph to extract relevant subgraphs G_{sub}. (3) **Knowledge Contextualization**: Prepare the extracted subgraphs in triplet format and combine them with the retrieved textual documents to form a comprehensive knowledge context $C_{hybrid} = C_{docs} \cup C_{graph}$. (4) **Multi-modal Answer Generation**: Provide both the retrieved documents $D_{retrieved}$ and serialized subgraphs C_{graph} to the LLM for answer generation, including the entity IDs. This hybrid approach aims to maximize knowledge coverage by integrating the reliability of curated knowledge graphs with the richness of textual documents.

3.4 Modular and Agentic RAG Scenarios

Scenario 6 - Modular RAG with well-defined workflow: This scenario serves as a baseline approach for comparison with subsequent Agentic RAG implementations. It employs a predetermined, sequential workflow to process queries. The modular components are executed in a sequence: (1) **Query Reformulation**: The input query q is reformulated into an optimized query q' to improve retrieval effectiveness. (2) **Vector Retrieval**: Perform similarity search in the vector index space of entity documents D to retrieve a set of candidate documents $D_{candidates} = \{d_1, d_2, \ldots, d_k\}$. (3) **Reranking**: Apply reranking algorithms to the retrieved candidates, producing an ordered list D_{ranked}. (4) **Answer Generation**: Generate the final answer as well as provide the list of selected entities $E_{selected}$ used to answer. This deterministic workflow provides a controlled baseline for evaluating the benefits of increased agent autonomy in subsequent scenarios.

Scenario 7 - Agentic RAG with modular components as tools: In this scenario, all modular components from Scenario 6 are transformed into tools available to an AI agent, which autonomously decides the workflow and tool utilization strategy. The available tool set includes: Query reformatter, Document retriever, and Reranker. The answer generation and entity selection are directly embedded into the agent's system prompt. The agentic workflow operates as follows: (1) **Autonomous Tool Selection**: Given query q, the agent selects appropriate tools from $\mathcal{T}$ based on its reasoning about the query requirements. (2) **Multi-step Execution**: The agent performs iterative tool calls $\{t_1, t_2, \ldots, t_n\}$ where $t_i \in \mathcal{T}$, with each step informed by previous results. (3) **Dynamic Query Composition**: Based on intermediate results R_{i-1}, the agent can compose new queries q_i for subsequent retrieval steps, enabling adaptive information gathering. (4) **Convergence Decision**: The agent determines when sufficient information has been gathered to provide a comprehensive answer. This approach enables flexible, context-aware workflows while maintaining access to specialized processing capabilities.

Scenario 8 - Autonomous agentic RAG with minimal tools: This scenario reduces tool availability to test the agent's ability to perform complex reasoning with minimal external support. The agent must internalize processing steps that were previously handled by dedicated tools. The agent configuration includes: (1) **Single Retrieval Tool**: The agent has access to only one tool $T_{retrieve}$ for vector index search over entity documents. (2) **Implicit Processing Instructions**: The system prompt instructs the agent to consider processes such as query reformulation, reranking, and entity selection, but provides no specific tools to perform these operations. (3) **Self-guided Reasoning**: The agent must internally implement logic for: Query analysis and reformulation strategies, Result evaluation and relevance assessment, Entity identification and selection criteria and Information synthesis and answer construction. (4) **Iterative Refinement**: The agent can perform multiple retrieval iterations, composing new queries q_i based on analysis of previous results R_{i-1}, enabling progressive information refinement. This scenario evaluates whether advanced language models can effectively internalize complex retrieval and reasoning processes without explicit tool support.

Scenario 9 - Autonomous agentic RAG with knowledge graph retrieval tool: This scenario extends Scenario 8 by providing the agent with a KG retrieval tool that is the same one as in Scenario 3 and 5. This scenario tests the agent's ability to orchestrate complementary knowledge sources.

3.5 Context Optimization

In our experiments with GraphRAG and Agentic RAG systems, we observed significant token usage and cost challenges that frequently approached or exceeded LLM context limits. Analysis revealed two primary sources of redundancy: (1) the conventional triplet format

(`entity1-relation-entity2`) of graph representation repeats entity names when pairs share multiple relations; (2) conventional agentic frameworks like Strands Agents store retrieval results flatly and inject entire session history into each subsequent LLM call, causing overlapping subgraphs and duplicate documents to accumulate in multi-steps.

To address these issues, we propose a three-fold context optimization strategy. First, we introduce a **relation-grouped graph representation** that transforms conventional triplets into compact format (`entity1 - (relation1|relation2|relation3) - entity2`), reducing token usage from $O(n)$ to $O(1)$ for entities connected by n relations. Second, we implement **graph retrieval deduplication** by maintaining a single unified subgraph throughout the agent session, merging new graph retrievals through entity-level and relation-level deduplication to ensure context grows sublinearly. Third, we implement content-aware **document retrieval deduplication** using content hashes to identify and eliminate duplicate documents or chunks before adding them to agent memory.

Beyond deduplication, we further reduce token overhead by modifying the agent's loop design and retrieval pattern. Rather than issuing one query per tool call (the default ReAct (Yao et al., 2023) behavior), we implement **batch agentic retrieval strategy**, combining ReAct and ReWOO (Xu et al., 2025) style which instructs the agent to formulate multiple complementary sub-queries and execute them in a single batched retrieval call (Algorithm 1). This hybrid ReAct-ReWOO approach reduces the number of LLM round-trips, each of which resends the full conversation history. See Appendix D for a detailed discussion of the batch strategy's design rationale and its interaction with the retrieval-generation gap.

Moreover, we further investigate the variants of extended retrieval by utilizing the saved token budget such as increasing the max paths and subgraphs, and retrieving nonredundant text chunks. These optimizations also enable controlled experiments to study the retrieval-generation gap, as token savings can be reinvested in expanded retrieval while monitoring whether generation quality follows.

Algorithm 1 Batch Agentic Retrieval (Hybrid ReAct-ReWOO Style)

Require: Query q, retrieval tool $\mathcal{T}$, max iterations K
Ensure: Answer a, entity IDs E

$\mathcal{M} \leftarrow \emptyset$ ▷ Agent memory
for $k = 1, \ldots, K$ **do** ▷ ReAct outer loop
  // Think: analyze query and current memory
  $\{q_1, q_2, \ldots, q_m\} \leftarrow \textsc{Plan}(q, \mathcal{M})$
  // Act: batched retrieval (ReWoo-style)
  $\mathcal{R}_{\text{batch}} \leftarrow \mathcal{T}(\{q_1, \ldots, q_m\})$ ▷ Single tool call
  // Deduplicate before adding to memory
  $\mathcal{R}_{\text{new}} \leftarrow \textsc{Dedup}(\mathcal{R}_{\text{batch}}, \mathcal{M})$
  $\mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{R}_{\text{new}}$
  // Observe: evaluate sufficiency
  if $\textsc{Sufficient}(q, \mathcal{M})$ **then**
    break
  end if
end for
$a, E \leftarrow \textsc{Generate}(q, \mathcal{M})$
return a, E

4 Experimental setup

Dataset and metrics: We evaluate our proposed RAG scenarios on a semi-structured knowledge base dataset: STaRK-Prime (Wu et al., 2024). It's a precision medicine inquiry dataset where textual documents are from multiple sources for about 129K entities such as disease, drug, protein and gene, and 8.1M KG relations are from PrimeKG (Chandak et al., 2023). We use the official test set of human-generated queries as well as the following evaluation metrics: Hit@1, Hit@5, Recall@20 (R@20), and Mean Reciprocal Rank (MRR). Importantly, unlike the original STaRK benchmark (Wu et al., 2024), which evaluates raw retrieval rankings, our metrics assess end-to-end generation quality: the LLM generates a natural language answer and returns a narrowed set of entity IDs, which are then evaluated against ground truth. For context optimization experiments, we include mean Token usage per query (#Tokens) and Recall of Retrieval (RoR) regarding ground-truth entity IDs coverage before generation utilization.

Predefined knowledge graph: To prepare the predefined KG for graph search, we extracted all the 129K entities and 8.1M relations from STaRK-PRIME dataset, and import them to Amazon Neptune Analytics Graph in openCypher format. Then we used LlamaIndex (Liu, 2022) to load the Neptune graph into a property graph index, and used CypherTemplateRetriever to create the knowledge graph retriever. For subgraph extraction, we set h

= 2 for h-hop neighbors following Xia et al. (Xia et al., 2024) and maximum of 100 paths to avoid exponentially increasing number of neighbor nodes and relations.

Document retrieval: The document retrievers are built using Amazon Bedrock Knowledge Bases. As the entity description documents have feasible token size, and to keep the completeness of the entity description, no chunking was performed. We used Amazon Titan Text Embedding v2 for embedding, Amazon OpenSearch Serverless as the vector store and information such as the entity type, name as the meta information of the index. For the experiment with the computed knowledge graph from the documents, we used Amazon Neptune Analytics as the vector store and the pre-defined GraphRAG process in Bedrock Knowledge Bases. For each retrieval call, 20 documents are retrieved.

Agentic RAG implementation: Besides the Document Retrievers and Predefined KG Retriever, we also implemented LLM-based Query Reformatter and Document Reranker for modular RAG (Scenario 6) and corresponding agentic RAG (Scenario 7) experiments. We used Strands Agents (AWS, 2025) as the framework for Agentic RAG implementation, with its default ReAct-style (Yao et al., 2023) reasoning loop if not specified, where the agent iteratively thinks about the query, acts by calling a tool, and observes the result before deciding the next step. In Scenario 7, the agent has access to Document Retriever, Query Reformatter and Document Reranker as the tools. While in Scenario 8, the agent only has access to the Document Retriever, but it is instructed in system prompt to consider the other processes autonomously. The system prompt can be found in Appendix Box A. Scenario 9 is similar to Scenario 8, but the agent has access to the Predefined KG Retriever as well.

LLM: Testing the performance of different LLMs is not the current focus of this work. Thus to fairly compare different RAG scenarios, Anthropic Claude 3.7 Sonnet was used as the core LLM for the LLM-based modules/tools and agents.

5 Experimental results

Regular RAG: Table 1 provides the performance for Regular RAG based Scenarios (Scenarios 1-2). Scenario 1, using only entity description documents with retrieval-only configuration, achieved modest performance with Hit@1 of 0.4404 and MRR of 0.5455. The addition of LLM answer generation and entity selection significantly improved results, boosting Hit@1 to 0.6147 and MRR to 0.6769. Scenario 2, incorporating both entity description and relations documents, showed consistent improvements across all metrics. The retrieval-only approach achieved Hit@1 of 0.5596 and MRR of 0.6647, while the enhanced version with LLM processing reached the highest performance among basic RAG methods with Hit@1 of 0.6972 and MRR of 0.7531. These results clearly demonstrate the effectiveness of incorporating the relation information in a RAG system, even it is conducted in a very simple way as defined in Scenario 2: only adding the 1-hop neighbors and relations to the end of the original documents. Notably, R@20 drops from 0.7154 to 0.6119 when moving from retrieval-only to LLM-selected evaluation, indicating the LLM selects fewer but more precise entities. This divergence between retrieval and generation metrics is a recurring pattern across all scenarios, motivating our generation-aware evaluation approach.

GraphRAG: As shown in the second section of Table 1, among 3 GraphRAG scenarios (Scenarios 3-5), Scenario 5 consistently outperformed the others. Scenario 3, using predefined knowledge graphs only, performed poorly across all metrics (Hit@1: 0.1376, MRR: 0.1542), which indicates the challenges of a RAG system purely relying on graph search without utilizing any semantic information. In contrast, the enhanced performance on Scenario 5 over Scenario 3 clearly demonstrated the benefit of utilizing the semantic information in GraphRAG. Scenario 4, which utilized the computed knowledge graph from entity documents, exhibited extreme inconsistency. The retrieval-only configuration failed completely on Hit@1 (0.0000) while achieving competitive R@20 (0.6340). The version with LLM answer generation and entity selection improved Hit@1 to 0.5596 but reduced overall effectiveness (MRR: 0.6162). The performance gap between Scenario 4 and 5 reflected the gap between the computed KG vs predefined KG. In Scenario 4, the computed KG was generated by the extracted entities and relations within the documents, which is challenging to be as accurate and comprehensive as the predefined KG. Moreover, using a low quality computed KG could negatively impact the final RAG performance, which can explain the performance gap between Scenario 4 and Scenario 1. The enhanced performance of Scenario 5 over Scenario 1 proves the benefit of leveraging the relation information from the pre-

Table 1: Evaluation results for different RAG scenarios.

Methods	Hit@1	Hit@5	R@20	MRR
Regular RAG				
Scenario 1: RAG with Entity Description Documents only				
- retrieval only (no LLM answer generation and entity selection)	0.4404	0.6697	0.7154	0.5455
- with LLM answer generation and entity selection	0.6147	0.7523	0.6119	0.6769
Scenario 2: RAG with Entity Description and Relations Documents				
- retrieval only (no LLM answer generation and entity selection)	0.5596	0.7798	**0.7874**	0.6647
- with LLM answer generation and entity selection	**0.6972**	**0.8257**	0.6728	**0.7531**
GraphRAG				
Scenario 3: GraphRAG with predefined KG only (max 100 paths)	0.1376	0.1835	0.1467	0.1542
Scenario 4: GraphRAG with vector search and computed KG from entity documents				
- retrieval only (no LLM answer generation and entity selection)	0.0000	0.1468	0.6340	0.1036
- with LLM answer generation and entity selection	0.5596	0.6972	0.5625	0.6162
Scenario 5: GraphRAG with vector search and predefined KG				
- max 100 paths, 5 entity subgraphs	0.6422	**0.7798**	**0.6353**	**0.7011**
- max 50 paths, 5 entity subgraphs	0.6239	0.7615	0.6061	0.6858
- max 100 paths, 20 entity subgraphs	**0.6514**	**0.7798**	0.6131	**0.7072**
- max 200 paths, 5 entity subgraphs	0.6147	0.7615	0.6177	0.6831
Modular and Agentic RAG				
Scenario 6: Modular RAG with well-defined workflow				
- doc retriever only (same as Scenario 1)	0.6147	0.7523	0.6119	0.6769
- doc retriever, reranker	0.5963	0.7706	0.7140	0.6749
- query reformatter, doc retriever, reranker	0.6239	0.8073	**0.7857**	0.7030
Scenario 7: Agentic RAG with modular components as tools				
- tools: query reformatter, doc retriever, reranker	0.6514	0.8165	0.7841	0.7235
Scenario 8: Autonomous Agentic RAG with minimal tools				
- tools: doc retriever	**0.6881**	**0.844**	0.7745	**0.7549**
- tools: doc retriever, enabled thinking	0.6296	0.8056	0.7643	0.7160
Scenario 9: Autonomous Agentic RAG with KG retrieval tools				
- tools: doc retriever, pre-defined KG retriever	0.6055	0.7890	0.7125	0.6834

Table 2: Evaluation results for methods with context optimization.

Methods	Hit@1	Hit@5	R@20	MRR	RoR	#Tokens
Scenario 5-Opt: GraphRAG with vector search and predefined KG + Context opt.						
- (baseline, no opt) max 100 paths, 5 entity subgraphs	0.6422	0.7798	**0.6353**	0.7011	77.5%	49.1K
- (with opt) max 100 paths, 5 entity subgraphs	0.6330	0.7890	0.6202	0.7003	77.5%	**23.0K(-53%)**
- (with opt) max 500 paths, 5 entity subgraphs	0.6422	**0.7982**	**0.6346**	0.7095	79.6%	29.0K(-41%)
- (with opt) max 100 paths, 20 entity subgraphs	**0.6606**	0.7890	0.6074	**0.7194**	79.2%	33.3K(-32%)
- (with opt) max 500 paths, 20 entity subgraphs	0.6055	0.7798	0.6283	0.6835	**83.5%**	54.3K(+11%)
Scenario 8-Opt: Autonomous Agentic RAG with minimal tools + Context opt.						
- (baseline, no opt) tools: doc retriever (20 docs)	**0.6881**	**0.844**	**0.7745**	**0.7549**	83.8%	237K
- (with opt) tools: doc retriever (20 docs)	0.6330	0.8260	0.7470	0.7220	81.0%	**192K(-19%)**
- (with opt) tools: doc retriever (20 nonredundant docs)	0.6060	0.7800	0.7160	0.6860	86.0%	286K(+21%)
- (with opt) tools: doc retriever (20 docs, batch queries)	0.6240	0.8070	0.7610	0.7080	88.0%	197K(-17%)
- (with opt) tools: doc retriever (20 nonredundant docs, batch queries)	0.5413	0.6972	0.6510	0.6004	**94.4%**	220K(-7.2%)
Scenario 9-Opt: Autonomous Agentic RAG with KG retrieval tools + Context opt.						
- (baseline, no opt) tools: doc retriever (20 docs), KG retriever (100 paths, 5 subgraphs)	0.6055	0.7890	0.7125	0.6834	84.2%	466K
- (with opt) tools: doc retriever (20 docs), KG retriever (100 paths, 5 subgraphs)	0.6240	0.8260	0.7850	0.7120	84.7%	352K(-24%)
- (with opt) tools: doc retriever (20 docs), KG retriever (500 paths, 5 subgraphs)	**0.6420**	**0.8350**	**0.8040**	**0.7320**	84.5%	272K(-42%)
- (with opt) tools: doc retriever (20 docs), KG retriever (100 paths, 5 subgraphs), batch queries	0.5321	0.7615	0.7603	0.6365	88.7%	264K(-43%)
- (with opt) tools: doc retriever (20 docs), KG retriever (500 paths, 5 subgraphs), batch queries	0.5229	0.7615	0.7330	0.6296	**89.1%**	**244K(-48%)**

Table 3: Comparison of LLM extracted vs. missed answer entities by position metrics.

Group	**In Doc Retrieval**	**Mean Rank**	**Med. Rank**	**Mean Tok #**	**Med. Tok #**	**Mean Tok %**	**Med. Tok %**
LLM Extracted	96.3%	5.3	3	1,888	376	10.5%	2.9%
LLM Missed	78.2%	61.0	11	5,534	5,003	36.8%	35.9%

defined KG. However, it is interesting to notice that Scenario 2 outperformed Scenario 5, indicating a simple integration of 1-hop relations could lead to better performance than a more sophisticated GraphRAG approach. This result could also due to the non-optimized subgraph representation (triplets) as well as the hyper-parameter selection (e.g., number of hops, maximum number of paths, number of entity for subgraphs). In Scenario 2, the 1-hop relations was grouped by relation types (see example in Figure 2 in Appendix), while in Scenario 5, the subgraphs were represented in triplet format, which often leads to lengthy context with which LLM could lose the attention due to the "lost in the middle" issue (Liu et al., 2023). Under Scenario 5, we compared different maximum number of paths and number of entity for subgraphs. It shows the trend of achieving better performance with higher number of entity for subgraphs and maximum number of paths. However, the performance could also drop due to the lengthy context and the "lost in the middle" issue.

Modular and Agentic RAG: For Modular RAG (Scenario 6), the performance improved while adding new component, which demonstrated the value of component specialization. The complete pipeline with query reformatter, document retriever, and reranker achieved exceptional performance (Hit@1: 0.6239, Hit@5: 0.8073, R@20: 0.7857 MRR: 0.703). Scenario 7 achieved better performance than Scenario 6 with the same modular components (query reformatter, document retriever, and reranker) as tools, which indicates the power of AI agent for self-planning and multi-step reasoning. More interestingly, the autonomous agentic RAG system (Scenarios 8) achieved the best performance under the Modular and Agentic RAG category, which indicates the potential for AI Agent to conduct complex tasks autonomously with minimal human support or predefined specialization tools. However, enabling thinking or adding the predefined KG retriever (Scenario 9) doesn't seem to help. In Appendix A, we shared one example of the thought process of the agent in Scenarios 8, outputted by Strands Agents. That example demonstrates the agent's ability to handle complex, multi-faceted queries requiring extensive domain knowledge integration while maintaining high accuracy. The agent's autonomous reasoning and adaptive behavior enabled it to successfully navigate the complexity of cross-domain medical and environmental relationships. However, the trace also reveals a critical limitation: after several retrieval calls, the accumulated session history repeatedly triggers context window overflow errors, forcing the agent into increasingly narrow searches and ultimately returning incomplete results. This highlights the need for the context optimization methods presented next.

Context Optimization: Table 2 demonstrates consistent token reduction across Scenarios 5, 8, and 9 with context optimization. For Scenario 5 (GraphRAG), optimization achieves 53% token reduction with comparable performance at baseline parameters, while expanding to 500 paths yields the best Hit@5 (0.7982) with 41% reduction. For Scenario 8, deduplication alone achieves 19% reduction with marginal trade-offs. Most impressively, Scenario 9 demonstrates that optimization benefits scale with retrieval complexity: the baseline configuration achieves 24% token reduction while improving Hit@5 and R@20, and expanding to 500 paths further enhances performance while achieving token reduction of 42%. This indicates that context optimization not only reduces overhead but also enhances agents' effective utilization of expanded hybrid text-graph retrievals.

The batch query variant (Algorithm 1) reduces tool calls while increasing retrieval coverage, as shown in both Scenario 8 and 9. It achieves the highest retrieval coverage (RoR: 94.4%) while still below baseline token usage when combined with nonredundant retrieval. These results demonstrate that batch agentic retrieval, which issues more speculative queries, is an effective approach for increasing retrieval coverage while reducing the number of LLM round-trips and thus the agent memory usage for repeated injection of session history. However, the expanded retrievals do not necessarily improve generation quality as shown in Table 2 across scenarios, directly evidencing the retrieval-generation gap discussed next.

Retrieval-Generation Gap: The performance gains with increasing retrieval scope remain below expectations across scenarios. Our deep-dive analysis (see Appendix B) reveals that while expanded retrieval significantly increases ground truth entity coverage, LLMs do not effectively leverage this additional information for answer generation, highlighting a critical gap between retrieval comprehensiveness and generation effectiveness.

To understand this gap, we compare LLM-extracted versus missed answer entities in Scenario 5-Opt (500 paths, 20 subgraphs), where retrieval coverage reaches 83.5% but LLM extraction only 47.9%. As shown in Table 3, extracted entities appear 3.5× earlier by token position (mean 10.5% vs. 36.8%) and the extraction rates drop sharply with position, indicating positional attention decay. Beyond position, qualitative analysis (Appendix C) identifies two additional factors: LLM preference for canonical entities over exhaustive enumeration, and query-induced cardinality expectations suppresses multi-entity extraction.

These findings suggest that current retrieval-oriented metrics (Hit@k, MRR) may overstate the benefit of expanded retrieval, and that evaluation of RAG systems should separately assess retrieval coverage and generation utilization. Besides, the study (Appendix B) also reveals that the agent with optimized context autonomously calls retrieval tool more, without explicit prompting. These observations merit further investigation.

6 Conclusions and future work

We introduce a comprehensive framework for evaluating Regular RAG, GraphRAG, Modular RAG, and Agentic RAG architectures on semi-structured knowledge bases. Our study demonstrates that the question "Is GraphRAG needed?" requires a nuanced, context-dependent answer. While GraphRAG shows promise for complex relational queries, simpler RAG variants can deliver competitive performance with lower overhead, whereas autonomous Agentic RAG with minimal specialized tools achieves superior results across the board. Our context optimization method achieves 19%-53% token reduction while maintaining or improving performance, demonstrating practical value for production deployment. Moreover, we identify a retrieval-generation gap that has implications for how RAG systems are evaluated. Our analysis shows that expanded retrieval does not translate to proportional generation improvements, driven by positional attention decay, semantic salience preferences, and query-induced cardinality effects. Future work should address the retrieval-generation gap, through context restructuring and prompt designs encouraging exhaustive extraction. Other directions could develop hybrid approaches balancing efficiency and adaptiveness, explore domain-specific graph utilization strategies.

Limitations

Our experiments are conducted on a single dataset (STaRK-Prime) from the precision medicine domain, which may limit generalizability to domains with different structural characteristics or relational complexity. However, the dataset's scale (129K entities, 8.1M relations) and its combination of textual and relational data make it representative of real-world semi-structured knowledge bases, and our framework is designed to be dataset-agnostic. Similarly, all experiments use a single LLM (Anthropic Claude 3.7 Sonnet). While relative scenario performance may shift with different models, our core findings (e.g., the retrieval-generation gap and the effectiveness of simple relational augmentation) reflect architectural properties rather than model-specific behaviors.

The agentic RAG scenarios exhibit inherent non-determinism in tool selection and planning, which can introduce variability despite using temperature zero. Our evaluation is limited to retrieval-oriented metrics (Hit@1, Hit@5, R@20, MRR) due to lack of ground truth natural language answers, and does not assess dimensions such as factual faithfulness or hallucination rates, which we leave for future work. Finally, we did not exhaustively tune hyperparameters across all scenarios, which may leave room for further performance improvements in individual configurations.

Ethics Statement

This work uses the publicly available STaRK-Prime dataset (Wu et al., 2024), derived from open biomedical knowledge bases including PrimeKG (Chandak et al., 2023). No human subjects were involved in our experiments. All evaluations were conducted using automated metrics on existing benchmark queries. Our experiments rely on commercial LLM APIs (Anthropic Claude via Amazon Bedrock), which incur computational costs and associated energy consumption. The biomedical domain application discussed in this work is intended for research evaluation purposes and should not be used for clinical decision-making without proper validation.

Acknowledgements

The authors would like to thank Gaurav Rele, Francisco Calderon Rodriguez, and Wan Chen for insightful discussions that helped shape this work. The authors are also grateful to Anila Joshi and Taimur Rashid for their guidance and continued support of this research.

References

AWS. 2025. Strands agents sdk: A new open-source framework for building agents. https://github.com/strands-agents/sdk-python/.

Payal Chandak, Kexin Huang, and Marinka Zitnik. 2023. Building a knowledge graph to enable precision medicine. *Scientific Data*, 10(1):67.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.

Yunfan Gao, Yun Xiong, Meng Wang, and Haofen Wang. 2024. Modular rag: Transforming rag systems into lego-like reconfigurable frameworks. *arXiv preprint arXiv:2407.21059*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Jerry Liu. 2022. Llamaindex: Data framework for llm-based applications. https://www.llamaindex.ai/.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*.

Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. 2024. Graph retrieval-augmented generation: A survey. *arXiv preprint arXiv:2408.08921*.

Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. 2025. Agentic retrieval-augmented generation: A survey on agentic rag. *arXiv preprint arXiv:2501.09136*.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. *arXiv preprint arXiv:1803.06643*.

Shirley Wu, Shiyu Zhao, Michihiro Yasunaga, Kexin Huang, Kaidi Cao, Qian Huang, Vassilis N Ioannidis, Karthik Subbian, James Zou, and Jure Leskovec. 2024. Stark: Benchmarking llm retrieval on textual and relational knowledge bases. *Advances in Neural Information Processing Systems*, 37:127129–127153.

Yu Xia, Junda Wu, Sungchul Kim, Tong Yu, Ryan A Rossi, Haoliang Wang, and Julian McAuley. 2024. Knowledge-aware query expansion with large language models for textual and relational retrieval. *arXiv preprint arXiv:2410.13765*.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Namber Muber, Eric Xie, and Xin Qi. 2025. Rewoo: Decoupling reasoning from observations for efficient augmented language models. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

A Agentic RAG Case Study

To illustrate the effectiveness of the autonomous agentic RAG approach, we examine a complex query "Why do prostate cancer cases rise in agriculturally intensive areas in California?" from the STaRK-Prime dataset. This query represents a challenging information retrieval task as it requires understanding multiple domains (oncology, agricultural science, and geographical epidemiology) and establishing causal relationships. Here in Box A is the log (thought process) of the agent as outputted by Strands Agents.

Box A: Case Study for Agentic RAG

Query: Why do prostate cancer cases rise in agriculturally intensive areas in California?
Agent:
I'll help you find relevant information about the relationship between prostate cancer cases and agriculturally intensive areas in California. Let me search the knowledge base for relevant documents.
Tool #1: retrieve
Let me search for more specific information about prostate cancer and agricultural areas in California.
Tool #2: retrieve
Let me search specifically for information about prostate cancer in California agricultural areas.
Tool #3: retrieve
Let me search specifically for prostate cancer in relation to agricultural areas.
Tool #4: retrieve
Let me search specifically for information about prostate cancer in California.
Tool #5: retrieve
Based on my searches, I need to look specifically for prostate cancer information.
Tool #6: retrieve
bedrock threw context window overflow error
Let me try a more focused search:
Tool #7: retrieve
bedrock threw context window overflow error
Let me try with a smaller result set:
Tool #8: retrieve
Let me try a more specific search for prostate cancer:
Tool #9: retrieve
bedrock threw context window overflow error
Let me try with a more focused search:
Tool #10: retrieve
bedrock threw context window overflow error
Let me try with a very specific search:
Tool #11: retrieve
Let me try one more search with a different approach:
Tool #12: retrieve
bedrock threw context window overflow error
Let me try with a very specific search:
Tool #13: retrieve
Based on my searches, I've found limited information specifically about prostate cancer in agricultural areas of California. Let me compile what I've found and provide the entity IDs in the required format.

As shown in Box A, the agent demonstrated sophisticated reasoning capabilities through multiple stages:

1. **Initial Query Decomposition:** The agent first analyzed the query's multiple components: (1) Geographic context (California); (2) Medical condition (prostate cancer); (3) Environmental factor (agricultural intensity); (4) Causal relationship ("why" and "rise")

2. **Dynamic Search Strategy:** The agent implemented an iterative search process, demonstrating: (1)Progressive refinement of search terms; (2) Context-aware query reformulation; (3)Autonomous decision-making about when to pursue additional information; (4) Self-correction when initial searches yielded insufficient results

3. **Adaptive Information Integration:** When faced with retrieval challenges, the agent automatically adjusted its approach by: (1) Increasing search specificity; (2) Decomposing complex relationships into manageable components; (3) Synthesizing information across multiple retrievals; (4) Maintaining context coherence across search iterations.

This example demonstrates the system's ability to handle complex, multi-faceted queries requiring extensive domain knowledge integration while maintaining high accuracy. The agent's autonomous reasoning and adaptive behavior enabled it to successfully navigate the complexity of cross-domain medical and environmental relationships. However, it also reveals a critical failure mode: repeated context window overflow errors (occurring in 6 of 13 tool calls), which motivates our context optimization work. This case study underscores the practical necessity of context management for agentic RAG deployment.

Box B: Agent System Prompt - Scenario 8

"""
You are an expert information retrieval agent specializing in scientific and medical knowledge.
<task>
Your task is to find relevant documents from a knowledge base using the 'retrieve' tool. Do NOT try to directly answer queries - focus on finding and returning relevant entity IDs.
</task>
<instructions>

1. Analyze the query and, if needed, reformulate it to improve search effectiveness.

2. Then use the retrieve tool to search the knowl-

edge base with the original or reformulated query.

3. Use the following infomation when calling the retrieve tool:

- knowledgeBaseId: {kd_id}
- numberOfResults: {num_results}
- score: 0.0

4. If you determine that the initial results are insufficient or that follow-up information would be beneficial:

- Formulate additional, more specific queries based on the initial results
- Perform follow-up retrievals using these new queries with the same retrieve tool and parameters
- Incorporate the new results into your analysis and reranking

5. Rerank all retrieved results based on their relevance to the original query, combining results from both initial and follow-up retrievals

6. Return the top UNIQUE entity IDs in the final reranked list. Each ID must come directly from the 'entity_id' field of the retrieved documents.

</instructions>
<output_format>
Your response must be wrapped in <json> tags and follow these requirements:

- Only use numeric entity IDs that appear in the 'entity_id' field of retrieved documents
- Do not use document names, titles, or other identifiers
- Entity IDs must be numbers, not text strings

<json>

```
{{

"final_answer": "final answer to the user
question" # The final answer should include
the 'entity_id' of the entities used to
answer the user question
"entity_ids_for_final_answer": [1234, 5678]
# Array of numeric entity IDs ordered by
relevance

}}
```

</json>
</output_format>
Return the output in JSON format wrapped in <json> tags as specified above. Do not include any additional text or explanations outside the <json> tags.
"""

B Deep-dive Analysis on Context Optimization Results

Table 4 exhibits the deep-dive analysis on the variants from Scenario 5-Opt (GraphRAG). It reveals a critical disconnect between retrieval comprehensiveness and generation effectiveness in GraphRAG systems. While document retrieval maintains consistent ground truth entity recall at 73.9% across all variants, expanding KG retrieval parameters dramatically improves entity coverage—from 54.9% (100 paths, 5 subgraphs) to 83.5% (500 paths, 20 subgraphs), representing a 52% relative improvement. However, this substantial increase in retrieved relevant entities does not translate to improved answer quality; LLM answer recall remains nearly flat between 45.4%-48.2%, and even slightly decreases with the most comprehensive retrieval configuration (20 subgraphs). This phenomenon suggests that LLMs struggle to effectively identify and synthesize relevant information from expanded graph contexts, potentially due to information overload, difficulty in distinguishing signal from noise in dense graph structures, or limitations in reasoning over large relational contexts. These findings highlight that simply retrieving more graph information is insufficient. Future work could address how to better present, rank, or guide LLMs in utilizing comprehensive graph retrievals for answer generation.

Table 5 exhibits the deep-dive analysis for Scenarios 8-Opt and 9-Opt. Token consumption scales primarily with sequential tool calls rather than retrieval volume, as each call resends the full conversation history. Batch strategies reduce tool calls substantially (from 4.2 to 2.8 for standard batch, and to 1.8 for nonredundant batch) while increasing retrieval coverage (88.0% and 94.4% respectively vs. 83.8% baseline), confirming that retrieval comprehensiveness does not require many LLM round-trips.

However, batch planning bypasses the intermediate think-act-observe reasoning of iterative ReAct, causing broader, more speculative retrieval. Per-query analysis reveals this directly: the nonredundant batch variant presents 219 documents (128K input tokens) at the final generation step versus 87 documents (67K tokens) for the iterative nonredundant variant. Despite achieving 94.4% retrieval coverage, the 2.5× larger generation context triggers the retrieval-generation gap, producing the lowest answer quality (Hit@1: 0.5413, LLM Answer: 71.1%). This demonstrates that batch retrieval's benefit lies in retrieval efficiency (fewer tool calls, higher coverage per token spent on session overhead), not in generation quality so far, as the final context size at the generation step is the critical

bottleneck.

This pattern intensifies in Scenario 9, where batch variants achieve the largest token reductions (43–48%) but the worst end-to-end metrics across all Table 2 configurations (Hit@1: 0.52-0.53), even underperforming the unoptimized baseline (0.6055). The combined text and graph retrievals from batch planning create an oversized context that the LLM cannot effectively process. In contrast, the non-batch variant (500 paths) achieves 42% reduction while improving all metrics over the baseline, confirming that iterative agent reasoning serves as a critical information filter between retrieval and generation.

Separating retrieval coverage from LLM utilization across all agentic variants reveals a consistent pattern: configurations with higher RoR do not produce higher LLM answer recall. In the test cases where both iterative and batch configurations retrieved the same answer entity, ranking position in the final output differed—with 45 favoring one variant and 58 favoring the other, despite identical retrieval and temperature=0. This variance in final answer selection, not retrieval, drives accuracy differences between configurations.

Furthermore, failure analysis identifies three structural limitations of vector search that neither agentic iteration nor graph augmentation can address: inverted relationships, off-label information buried in nested relation fields, and queries requiring set intersection across structured attributes. Graph tools provided marginal unique retrieval (+9 answers) but degraded reranking quality—the additional context diluted relevance signals, causing answers to rank lower in final output.

C Retrieval-Generation Gap: Position Analysis

To understand why LLMs fail to utilize retrieved entities, we analyze the gap between retrieval coverage (83.5%) and LLM extraction (47.9%) in Scenario 5-Opt (500 paths, 20 subgraphs). Of 237 answer entities present in the prompt, the LLM extracts only 136 (57.4%), missing 101. The prompt structure places retrievals from documents (up to 20 entities with descriptions) first, followed by graph context in triplet format, with an average of 538 unique entities and 14,862 tokens per prompt.

Table 3 compares extracted versus missed entities across position metrics. Extracted entities appear 3.5× earlier by token position (mean 10.5% vs 36.8%, or 1,888 vs 5,534 tokens) and are more likely to appear in the Doc retrieval section (96.3% vs 78.2%).

Table 6 shows extraction rates by token position decile. Entities in the first 10% of tokens achieve an 85.5% hit rate, dropping sharply to 26.3% at 30%–40% and 0% at 70%–80% and 90%–100%. This is consistent with the "lost in the middle" phenomenon (Liu et al., 2023).

Table 7 shows extraction rates by absolute entity rank. Rank-1 entities achieve 97.9% hit rate, but this drops to 43.5% for ranks 6–10 and 0% beyond rank 50. All 136 extracted entities come from the top 50; entities appearing only in the graph triplet section have zero extraction rate.

Case Study: Gum Disease Query. To illustrate the interplay between position and semantic relevance, we examine the query: *"I have inflammation in my gums, and it turns swollen and puffy. Which disease could potentially be the issue?"* The ground truth contains 21 disease entities; 18 were retrieved but only 4 were extracted by the LLM. Table 8 shows the retrieved answer entities ordered by rank.

The LLM extracted the top 3 entities plus "periodontitis" at rank 7 (token position 35.0%), while missing three subtypes at earlier positions (ranks 4–6, token positions 17.6%–29.4%). This reveals that **semantic relevance acts as a secondary factor**: the LLM prefers canonical, well-known terms (e.g., "periodontitis") over specialist subtypes (e.g., "suppurative periapical periodontitis"), behaving more like a clinician giving a differential diagnosis than exhaustively listing all matches. The ground truth expects exhaustive enumeration, creating a systematic mismatch with LLM generation behavior.

Case Study: Hypertension Drug Query. We examine a second query: *"Imagine you're a pharmacologist on a quest to find an experimental drug to treat high blood pressure. What is an investigational adrenergic alpha-1 receptor antagonist that has been studied for the treatment of hypertension?"* All 14 ground truth entities were successfully retrieved into the prompt. However, the LLM extracted only 1 entity (Bunazosin, token position 11.1%), despite several valid answers appearing very early in the graph section (e.g., Doxazosin at 0.4%, Guanadrel at 2.8%, Terazosin at 4.4%). The early positions of the missed entities contradict a purely position-based explanation. Instead, the singular phrasing "What is an investi-

Table 4: Deep-dive analysis for different variants of Scenario 5-Opt.

Methods	Ground Truth Entity Recall			
	Doc Retrieval	KG Retrieval	Doc+KG Retrieval	LLM Answer
Scenario 5-Opt: GraphRAG with vector search and predefined KG + Context opt.				
- (with opt) max 100 paths, 5 entity subgraphs	73.9%	54.9%	77.5%	48.2%
- (with opt) max 200 paths, 5 entity subgraphs	73.9%	58.1%	77.8%	48.2%
- (with opt) max 500 paths, 5 entity subgraphs	73.9%	62.7%	79.6%	47.9%
- (with opt) max 100 paths, 20 entity subgraphs	73.9%	79.2%	79.2%	45.4%
- (with opt) max 500 paths, 20 entity subgraphs	73.9%	83.5%	83.5%	47.9%

Table 5: Deep-dive analysis for different variants of Scenario 8-Opt and Scenario 9-Opt.

Methods	#Tool Calls	Latency	Ground Truth Entity Recall	
			Doc+KG Retrieval	LLM Answer
Scenario 8-Opt: Autonomous Agentic RAG with minimal tools + Context opt.				
- (baseline, no opt) tools: doc retriever (20 docs)	4.2	27.4s	83.8%	77.4%
- (with opt) tools: doc retriever (20 docs)	5.3	27.6s	81.0%	74.7%
- (with opt) tools: doc retriever (20 nonredundant docs)	4.9	31.3s	86.0%	71.6%
- (with opt) tools: doc retriever (20 docs, batch queries)	2.8	22.8s	88.0%	76.1%
- (with opt) tools: doc retriever (20 nonredundant docs, batch queries)	1.8	19.8s	94.4%	71.1%
Scenario 9-Opt: Autonomous Agentic RAG with KG retrieval tools + Context opt.				
- (baseline, no opt) tools: doc retriever (20 docs), KG retriever (100 paths, 5 subgraphs)	6.7	71.8s	84.2%	71.2%
- (with opt) tools: doc retriever (20 docs), KG retriever (100 paths, 5 subgraphs)	8.4	43.7s	84.7%	78.6%
- (with opt) tools: doc retriever (20 docs), KG retriever (500 paths, 5 subgraphs)	7.2	35.8s	84.5%	76.4%
- (with opt) tools: doc retriever (20 docs), KG retriever (100 paths, 5 subgraphs), batch queries	2.4	39.8s	88.7%	75.7%
- (with opt) tools: doc retriever (20 docs), KG retriever (500 paths, 5 subgraphs), batch queries	2.3	30.4s	89.1%	73.6%

Table 6: LLM extraction rate by entity token position decile.

Token Position	Hits	Misses	Hit Rate
0%–10%	94	16	**85.5%**
10%–20%	14	16	46.7%
20%–30%	14	12	53.8%
30%–40%	5	14	26.3%
40%–50%	3	13	18.8%
50%–60%	4	10	28.6%
60%–70%	1	8	11.1%
70%–80%	0	5	0.0%
80%–90%	1	4	20.0%
90%–100%	0	3	0.0%

Table 7: LLM extraction rate by absolute entity rank. The anomalous 62.5% at ranks 21–50 reflects only 8 samples.

Rank	Hits	Misses	Hit Rate
1	47	1	**97.9%**
2	20	9	69.0%
3	16	4	80.0%
4–5	16	10	61.5%
6–10	20	26	43.5%
11–20	12	29	29.3%
21–50	5	3	62.5%
51+	0	19	**0.0%**

gational...antagonist" appears to induce a **single-answer cardinality expectation**: the LLM interprets the query as requesting one entity rather than an exhaustive list, producing only one answer even when many valid ones are present in the high-attention zone. This contrasts with the gum disease query's plural phrasing ("Which disease..."), where the LLM extracted multiple entities. This case reveals a third factor in the retrieval-generation gap beyond position and semantic relevance: the query's grammatical framing can systematically suppress multi-entity extraction.

These findings suggest the retrieval-generation gap stems from three interacting factors: (1) positional attention decay in long contexts, (2) LLM preference for semantically salient entities over exhaustive enumeration, and (3) query-induced cardinality expectations—where singular phrasing (e.g., "What is...") causes the LLM to output a single entity even when multiple valid answers exist in the context. Simply expanding retrieval scope is insufficient; future work should investigate how to structure and present retrieved information to maximize LLM utilization.

D Batch Agentic Retrieval Strategy

In our analysis of agentic RAG token consumption, we identified that the dominant cost driver is sequential tool calls rather than retrieval volume: each call resends the full conversation history to the LLM, so a 5-call session consumes approximately 250K tokens while a 3-call session uses only 36K regardless of documents retrieved per call. This observation motivates a hybrid retrieval strategy that combines elements of ReAct (Yao et al., 2023) and ReWOO (Xu et al., 2025).

Scenario 1

- entity_id: 10022
- name: RDH5
- type: gene/protein
- source: NCBI
- details:
  - query: RDH5
  - alias (other gene names): ['9cRDH', 'HSD17B9', 'RDH1', 'SDR9C5']
  - genomic_pos (genomic position): {'chr': '12', 'end': 55724705, 'ensemblgene': 'ENSG00000135437', 'start': 55720367, 'strand': 1}
  - name (gene name): retinol dehydrogenase 5
  - summary (protein summary text): This gene encodes an enzyme belonging to the short-chain dehydrogenases/reductases (SDR) family. This retinol dehydrogenase functions to catalyze the final step in the biosynthesis of 11-cis retinaldehyde, which is the universal chromophore of visual pigments. Mutations in this gene cause autosomal recessive fundus albipunctatus, a rare form of night blindness that is characterized by a delay in the regeneration of cone and rod photopigments. Alternative splicing results in multiple transcript variants. Read-through transcription also exists between this gene and the neighboring upstream BLOC1S1 (biogenesis of lysosomal organelles complex-1, subunit 1) gene. [provided by RefSeq, Dec 2010].

Scenario 2

- entity_id: 10022
- name: RDH5
- type: gene/protein
- source: NCBI
- details:
  - query: RDH5
  - alias (other gene names): ['9cRDH', 'HSD17B9', 'RDH1', 'SDR9C5']
  - genomic_pos (genomic position): {'chr': '12', 'end': 55724705, 'ensemblgene': 'ENSG00000135437', 'start': 55720367, 'strand': 1}
  - name (gene name): retinol dehydrogenase 5
  - summary (protein summary text): This gene encodes an enzyme belonging to the short-chain dehydrogenases/reductases (SDR) family. This retinol dehydrogenase functions to catalyze the final step in the biosynthesis of 11-cis retinaldehyde, which is the universal chromophore of visual pigments. Mutations in this gene cause autosomal recessive fundus albipunctatus, a rare form of night blindness that is characterized by a delay in the regeneration of cone and rod photopigments. Alternative splicing results in multiple transcript variants. Read-through transcription also exists between this gene and the neighboring upstream BLOC1S1 (biogenesis of lysosomal organelles complex-1, subunit 1) gene. [provided by RefSeq, Dec 2010].
- relations:
  ppi: {gene/protein: (ORC4, RGR, RLBP1, GNPNAT1, NME2P1, DDI2),}
  target: {drug: (Vitamin A, NADH),}
  associated_with: {disease: (congenital stationary night blindness, myopia, fundus albipunctatus, retinitis punctata albescens),}
  interacts_with: {cellular_component: (integral component of membrane, endoplasmic reticulum membrane, endoplasmic reticulum lumen),molecular_function: (androsterone dehydrogenase activity, NAD-retinol dehydrogenase activity, androstan-3-alpha,17-beta-diol dehydrogenase activity, protein homodimerization activity),pathway: (Retinoid cycle disease events, The canonical retinoid cycle in rods (twilight vision), RA biosynthesis pathway),biological_process: (steroid metabolic process, retinoid metabolic process, response to stimulus, visual perception, retinol metabolic process),}
  expression_present: {anatomy: (zone of skin, lymph node, adult mammalian kidney, intestine, blood, prefrontal cortex, anatomical system, testis, stomach, heart, brain, cerebral cortex, adipose tissue, esophagus, saliva-secreting gland, skeletal muscle tissue, colon, cortex of kidney, urinary bladder, pancreas, endometrium, myometrium, tibial nerve, muscle of leg, frontal cortex, temporal lobe, placenta, cerebellum, thyroid gland, lung, heart left ventricle, spleen, liver, small intestine, kidney, cerebellar cortex, prostate gland, adrenal gland, muscle tissue, fallopian tube, thoracic mammary gland, dorsolateral prefrontal cortex, multi-cellular organism, female gonad),}

Figure 2: Example of the documents used in Scenario 1 and 2.

Table 8: Retrieved answer entities for gum disease query, ordered by rank. ✓ = extracted by LLM, ✗ = missed.

Status	Entity (ID)	Rank	Tok %
✓	chronic gingivitis (98905)	1	0.2%
✓	gingivitis (38078)	2	6.0%
✓	necrot. ulcer. gingivitis (83971)	3	11.6%
✗	periodontitis, aggressive (27365)	4	17.6%
✗	periapical periodontitis (36107)	5	23.7%
✗	suppur. periapical period. (97566)	6	29.4%
✓	periodontitis (35985)	7	35.0%
✗	periodontal disease (36476)	8	40.6%
✗	periodontitis, chronic (28140)	9	41.2%
✗	stomatitis (36702)	10	46.8%
✗	apical periodontitis (95458)	11	47.2%
✗	gingival disease (36118)	12	52.6%
✗	gingival hypertrophy (95134)	13	52.8%
✗	pericoronitis (97540)	15	53.8%
✗	gingival cancer (37557)	19	55.1%
✗	epulis (39168)	20	68.4%
✗	acute pericementitis (95367)	151	81.4%
✗	herpes simplex gingivost. (97159)	229	88.7%

The default agent implementation in Strands Agents follows a standard ReAct-style loop: the agent *thinks* about the query, *acts* by calling a single tool with a single query, *observes* the result, and repeats. In contrast, our batch variant (Algorithm 1) modifies the inner action step: instead of issuing one retrieval query per tool call, the agent formulates multiple sub-queries simultaneously and executes them in a single batched tool call. The outer loop remains ReAct-style (iterative think-act-observe with a sufficiency check), while the inner action step follows the ReWOO principle of planning multiple retrieval actions before execution.

Concretely, at each iteration the agent is instructed to decompose the information need into multiple complementary sub-queries (e.g., targeting different entities, relations, or constraints mentioned in the original query). These sub-queries are passed to the retrieval tool in a single call, and the results are deduplicated using the content-aware hashing described in Section 3.5 before being added to the agent's memory. This reduces the number of LLM round-trips from an average of 4.2 sequential calls to 2.8 batched calls, while *increasing* retrieval coverage from 84% to 88% (Table 5), demonstrating that retrieval comprehensiveness does not require many LLM round-trips.

This design reduces the number of LLM round-trips, each of which resends the full conversation history (Appendix B). However, batch planning bypasses the intermediate reasoning of iterative ReAct: rather than evaluating and narrowing queries after each retrieval, all sub-queries are issued upfront based on initial analysis alone. This produces broader, more speculative retrieval that increases coverage but inflates the final generation context. When combined with nonredundant deduplication or multi-source retrieval (Scenario 9), the accumulated context exceeds the LLM's effective attention span, widening the retrieval-generation gap despite higher retrieval coverage (see Appendix B for detailed analysis).