SAGE: Structure Aware Graph Expansion for Retrieval of Heterogeneous Data

Anonymous ACL submission

Abstract

Retrieval-augmented question answering over heterogeneous corpora often requires evidence scattered across text, tables, and graph nodes. Existing approaches either rely on costly entity-level knowledge graphs or perform flat similarity search over independently chunked documents, limiting multi-hop retrieval across modalities. We propose SAGE (**S**tructure **A**ware **G**raph **E**xpansion), a framework that constructs a chunk-level graph offline using metadata-aware semantic links and retrieves evidence online by expanding neighborhoods from an initial set of retrieved chunks and reranking them with hybrid dense–sparse retrieval. We instantiate the seed retriever with hybrid retrieval for implicit cross-modal corpora and SPARK (**S**tructure **A**ware **P**lanning **A**gent for **R**etrieval over **K**nowledge **G**raphs) for explicit schema graphs. Experiments on OTT-QA and STaRK show consistent gains in retrieval recall, improving over strong baselines by 5.7 and 8.5 points, respectively.

1 Introduction

Retrieval-augmented generation (RAG) (Lewis et al., 2020) grounds large language models in external knowledge and has become a standard paradigm for question answering over domain-specific corpora. Most RAG systems rely on flat retrieval, where independently indexed text chunks are retrieved using sparse, dense, or hybrid similarity search (Mandikal and Mooney, 2024; Arabzadeh et al., 2021; Zhang et al., 2024; Chen et al., 2022). While effective for simple text-centric queries, this design struggles with complex reasoning over heterogeneous data that includes documents, tables, and structured metadata.

Flat retrieval treats each chunk independently, ignoring structural dependencies that often connect relevant evidence across documents or modalities. As a result, intermediate evidence that is weakly related to the query in embedding space but structurally connected to relevant content is frequently missed. For example, answering a question about a television show starring Trevor Eyster and its source book requires linking cast information, adaptation metadata, and author details, connections that are unlikely to be recovered through similarity search alone.

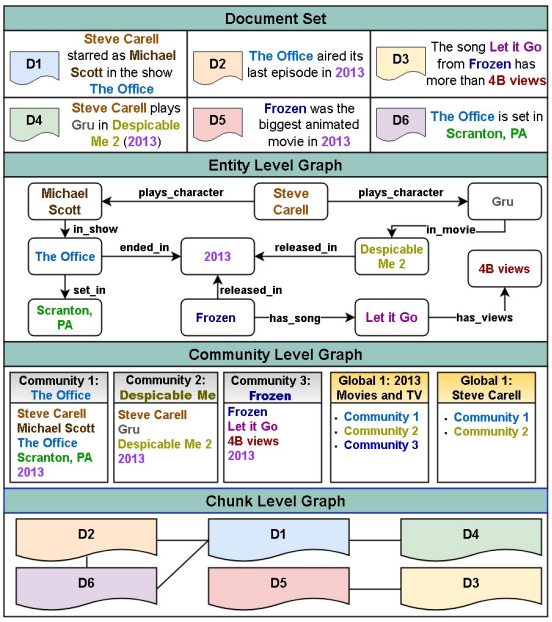

Figure 1: Multi-granular indexing strategies contrasting entity, community, and document-chunk graph structures.

To address these limitations, recent work augments RAG with graph-based representations that model relationships across heterogeneous data and enable multi-hop retrieval (Edge et al., 2024a; Jimenez Gutierrez et al., 2024; Mavromatis and Karypis, 2024). Prior approaches fall into two categories: *knowledge graphs* (KGs) and *similarity graphs*. KGs encode structured knowledge as typed entity-relation triples and support precise symbolic reasoning via traversal or query languages such as Cypher (Francis et al., 2018), and SPARQL (201, 2013), but require costly entity extraction and schema alignment, making them suitable for do-

Code available at `https://anonymous.4open.science/r/SAGE-MAY-ARR/README.md`

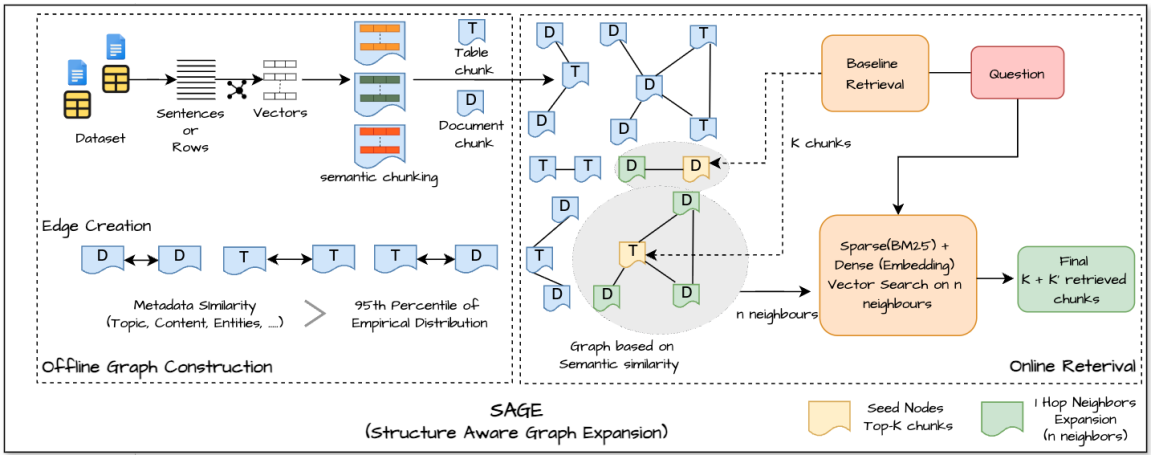

Figure 2: Overview of SAGE. **Offline:** we semantically chunk documents and tables into nodes and create edges using metadata-driven similarity (e.g., title, topic, content, entities) with percentile-based pruning. **Online:** a baseline retriever returns k seed nodes, we expand to n first-hop neighbors in the offline graph, and re-rank to select k' additional nodes, yielding a final context of $k+k'$.

mains with stable schemas. Similarity graphs represent text chunks, table segments, or semi-structured nodes with edges induced by embedding similarity, naturally supporting heterogeneous data without explicit entity extraction.

However, their effectiveness depends critically on edge construction: overly dense graphs introduce spurious neighbors, while overly sparse graphs hinder multi-hop reasoning. Figure 1 shows how node granularity impacts expressiveness, connectivity, and computational cost. **Entity-level graphs** capture fine-grained entities, supporting factoid queries but struggling with contextual or aggregate questions.**Community-level graphs** group entities or documents into clusters, improving scalability but reducing detail. These trade-offs motivate intermediate representations that balance granularity, scalability, and retrieval effectiveness.

We address this gap with a chunk-level graph that improves recall through structure-aware expansion over coherent text and table segments. Rather than retrieving many independent chunks, we select seed nodes and expand them via graph traversal, uncovering evidence linked through document structure, metadata, or shared context. Built offline and retriever-agnostic, the graph enables efficient multi-hop evidence aggregation across heterogeneous corpora. **Our contributions are as follows:**

- We introduce SAGE, a **structure-aware retrieval framework** that augments flat retrieval with chunk-level graph expansion, enabling neighbor-aware reasoning while offering a practical path when full KGs are infeasible but structural signals remain in metadata.
- Through **similarity-induced graph expansion**, SAGE consistently improves recall and QA accuracy over strong baselines.
- For explicit schema graphs, we further propose SPARK, a retrieval agent that performs **schema-constrained expansion**, outperforming naive dense expansion.

2 SAGE Approach

Given a corpus represented as a graph $G = (V, E)$ with node content $\text{text}(v)$ and optional metadata $\text{meta}(v)$, the retrieval goal for a query q is to return a ranked list of evidence nodes $R_k(q)$ such that relevant evidence appears in the top-k positions.

2.1 Offline Graph Construction

We construct heterogeneous graphs G over document chunks and table segments (for document graphs) or preserve native entity-relation schemas (for KGs). The graph construction process is dataset-agnostic and occurs offline before query time.

A. Data Processing and Node Creation

1. Semantic chunking of documents. We segment each document into sentences and embed overlapping context windows $w_i = [s_{i-1}, s_i, s_{i+1}]$ using a sentence encoder (Reimers and Gurevych, 2019). Let $\mathbf{h}_i$ denote the embedding of w_i. We compute cosine similarity between adjacent windows and place chunk boundaries at low-coherence points, defined as similarities below the 20th percentile of all adjacent-window similarities. Unlike fixed line- or token-based splitting, this semantic segmentation preserves topical continuity and produces more coherent chunks. If a document exhibits no significant similarity drops, indicating overall semantic

coherence, we instead split it into chunks of 4–6 lines. We then use an LLM to generate chunk metadata (topic, title, and entities) for downstream graph construction. We provide an example of this process in Appendix A

2. Table segmentation. We use an LLM to generate a short table description and column descriptions; these become part of the table-chunk metadata. For each table, we retain table title/description, column headers, and column descriptions, and 5–10 rows per segment to preserve vertical context while keeping segments within a bounded context size.

3. Semi-structured nodes. For semi-structured nodes, we treat each object as a "chunk" and construct metadata from its textual fields (e.g., name/title and long-form descriptions). These metadata fields are used to build downstream graph connectivity. We discuss metadata quality control, hidden LLM costs, and safeguards against generic generations in Appendix E.1.

B. Edge creation

Graph construction via metadata similarities. We embed these metadata fields using a Sentence-Transformer encoder and connect nodes based on modality-specific similarity signals: (1) **Document–Document** edges capture topic-topic similarity, content-content similarity, and entity overlap; (2) **Table–Table** edges capture column-column similarity, title-title similarity, and entity overlap; (3) **Table–Document** edges capture content-column similarity, topic-title similarity, and entity overlap; (4) **Semi-Structured Chunk** edges connect objects via similarity over their descriptive metadata.

We only keep a similarity edge when its metadata similarity exceeds the 95th percentile of the empirical distribution, which maintains sparsity and reduces dense-neighborhood noise. For large corpora, this threshold can be estimated using ANN-based candidate neighborhoods; further analysis is provided in the Appendix. We also preserve structural links (e.g., chunks from the same source) and retain explicit schema edges when available. Figure 2 illustrates the resulting graph structure.

2.2 Online Retrieval

At query time, we employ a two-stage retrieval strategy: (i) **initial baseline retrieval** to obtain k seed nodes, and (ii) **graph-based neighbor expansion and pruning** to select k' additional nodes. The baseline retrieval method differs between document graphs and KGs due to their distinct structural properties: hybrid sparse-dense retrieval works well for OTT-QA's text/table data, while SPARK is necessary for STaRK's schema-rich KGs (as hybrid retrieval performs poorly on STaRK, shown in Table 9).

A. Initial baseline retrieval.

1. Similarity graphs For document graphs (tables and text chunks), we use a hybrid sparse-dense retrieval baseline. We compute BM25 (Robertson et al., 2009) scores for lexical matching and cosine similarity over dense embeddings for semantic matching, then combine these scores to select the top-k seed nodes. This approach works well when chunks contain sufficient textual content and when relevance can be determined from surface-level semantic similarity.

2. Semi-Structured Knowledge Bases (SKBs) For semi-structured knowledge bases with explicit schemas (i.e., graphs whose nodes are associated with textual fields), BM25 and cosine similarity are limited. They ignore edge types and typed relations, cannot reason over multi-hop paths, and exhibit type bias, often retrieving nodes of the same type rather than the correct entities required by the query.

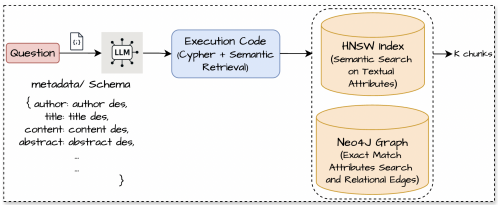

Figure 3: SKB baseline initial retrieval: An LLM agent generates a retrieval plan interleaving HNSW semantic search with Cypher symbolic queries.

To address this, we introduce SPARK, an LLM-based agent for SKB retrieval. Given a question, the agent accesses entity schemas, relations, and two retrieval tools, and generates a multi-step plan alternating between: **(1) Semantic Candidate Generation**: retrieving nodes via HNSW (Malkov and Yashunin, 2018) based on textual similarity and attributes; and **(2) Structural Expansion**: traversing edges with Cypher queries in Neo4j (Neo4j, Inc., 2026), constrained by entity types and relations to enforce multi-hop correctness.

We execute the generated plan and return the top-k retrieved nodes as seeds. A fallback to BM25 and cosine similarity is used when the plan returns no

results to ensure robustness, as the agent may fail to generate correct multi-step queries for certain questions. This combination balances the precision of schema-aware traversal with the coverage of traditional retrieval methods.

B. Graph-based neighbor expansion and pruning. Regardless of the baseline, we apply the same graph expansion operator over both document and KG graphs. SPARK acts only as the seed retriever for schema-rich KGs, while SAGE is retriever-agnostic. Given k seed nodes, we (1) collect first-hop neighbors, (2) rank candidates using BM25 and dense similarity to the query and retain the top-k', and (3) return the union of seeds and selected neighbors. We set $k' = k$, yielding a fixed $2k$ context size.

Because expansion is query-conditioned and followed by reranking, only relevant neighbors are retained, reducing topic drift. By leveraging precomputed graph edges instead of runtime pairwise similarity computations, SAGE enables efficient graph-aware retrieval. Our contribution lies in the graph construction and granularity rather than learned traversal or reranking policies.

3 Experiments

Datasets. **OTT-QA** (Chen et al., 2020) combines Wikipedia tables and passages. We use the dev split and reconstruct the graph from scratch, rebuilding all links rather than using the predefined connections. **STaRK** (Wu et al., 2024b) provides semi-structured knowledge graphs from the AMAZON, MAG, and PRIME domains. Synthetic queries are generated using relational templates and LLM-extracted properties, while human queries reflect more realistic language and ambiguity. Table 1 summarizes the resulting graph statistics.

Dataset	#Nodes	#Node types	#Edges	#Edge Types
OTT-QA	22,886	2	751,836	3
		STaRK		
AMAZON	13,920,204	2	40,876,246	4
MAG	1,872,968	4	36,853,636	5
PRIME	129,375	10	9,825,282	19

Table 1: Overview of datasets used in our experiments.

The two benchmarks target distinct settings. OTT-QA evaluates building a graph from unstructured and semi-structured data, testing whether semantic linking can recover useful structure without predefined relations. STaRK evaluates enhancing an existing knowledge graph, isolating the contribution of semantic similarity edges and their ability to complement explicit relations with additional retrieval and reasoning paths. Further details on node and edge types are in Appendix B .

Baselines for OTT-QA We compare against three retrieval baselines: (1) **GraphRAG Local Search** (Edge et al., 2024b), which retrieves an entity and its immediate graph neighborhood; (2) **Community Expansion**, which expands retrieved chunks using associated communities ($n = 3$ also includes the two nearest communities via embeddings); and (3) **Hybrid Search**, which combines BM25 and dense retrieval scores. For QA, we evaluate answer generation using the top-k retrieved documents ($k \in 2, 5, 10, 20$), as larger contexts approach practical limits. We also report **GraphRAG DRIFT Search**, which performs iterative graph traversal; since retrieval depth is adaptive, Recall@k is not directly applicable, and DRIFT is evaluated only in the end-to-end QA setting. Further details are in Appendix C.

Baselines for STaRK We compare our method with the following retrieval baselines: 1) Dense retrievers: (**ada-002**, **multi-ada-002**, **DPR** (Karpukhin et al., 2020)). 2) Sparse retriever: (**BM25**). 3) Hybrid retrievers: (**BM25+Dense**, **4StepFocus** (Boer et al., 2024), **FocusedR** (Boer et al., 2025)). 4) Graph-based retrieval: (**QA-GNN** (Yasunaga et al., 2021)). 5) Query expansion methods: (**HyDE** (Gao et al., 2023), **AGR** (Chen et al., 2024), **RAR** (Shen et al., 2024), **KAR** (Xia et al., 2024)). 6) Agentic systems: (**ReACT** (Yao et al., 2022), **Reflexion** (Shinn et al., 2023)). 7) Fine-tuned retrievers: (**mFAR** (Li et al., 2024a), **MoR** (Lei et al., 2025), **AvaTaR** (Wu et al., 2024a)). Further details on these baselines are in Appendix D.

Evaluation Metrics. For **OTT-QA**, we report both Exact Match (EM) for QA accuracy and Recall@k for retrieval performance. For **STaRK**, which does not include a downstream QA task, we follow the original evaluation protocol and report Recall@20 as the primary metric, along with Hit@1, Hit@5, and MRR.

Given the large number of gold evidence nodes per query in STaRK (e.g., ∼18 in the Amazon synthetic subset), we use Recall@20 as the primary metric. Unlike ranking-focused metrics such as Hit@k and MRR, which emphasize early retrieval quality, Recall@20 more accurately reflects overall evidence coverage and whether relevant supporting information is retrieved. We therefore focus on recall-based evaluation while also reporting Hit@1, Hit@5, and MRR for completeness and

transparency.

3.1 OTT-QA: Hybrid Text-Table Results

Baseline	Recall@k					
	2	5	10	20	50	100
GraphRAG	16.41	24.64	36.44	43.13	44.30	44.30
Community(n=1)	18.76	30.52	38.02	45.48	53.48	58.82
Community(n=3)	**33.54**	37.15	50.20	59.63	64.65	66.18
Hybrid Search	21.50	35.50	46.10	56.70	67.70	75.30
SAGE	24.90	**41.20**	**51.00**	**61.20**	**71.90**	**78.10**

Table 2: Recall@K comparison across different retrieval baselines. Higher values indicate better retrieval performance.

Graph-based retrieval improves Recall@k across all retrieval sizes. SAGE consistently outperforms all baselines across retrieval budgets (Table 2), achieving 61.2 Recall@20 and 78.1 Recall@100, compared to 56.7 and 75.3 for Hybrid Search. GraphRAG quickly plateaus because retrieval is confined to entity-level neighborhoods around the initial seeds, while Community Expansion remains limited by community boundaries. By leveraging document-level graph connections, SAGE captures broader context and retrieves more relevant evidence, yielding the strongest overall performance.

Baseline	Exact Match (EM, %)			
	2	5	10	20
GraphRAG (Local)	22.02	29.16	32.64	38.64
Community($n = 1$)	22.09	32.46	34.82	38.82
Community ($n = 3$)	**29.12**	34.18	39.58	52.67
Hybrid Search	21.83	33.81	37.06	43.05
SAGE	27.21	**35.94**	**42.31**	**55.79**

Table 3: QA accuracy comparison across retrieval baselines.

End-to-end QA accuracy and efficiency. SAGE achieves the best QA performance across nearly all retrieval budgets (Table 3), reaching 42.3% EM at $k = 10$ and 55.8% at $k = 20$. Performance largely follows the Recall@k ranking, with stronger retrievers yielding better downstream QA accuracy. Consistent with their high recall, Community Expansion ($n = 3$) and SAGE achieve the strongest QA results, while GraphRAG DRIFT attains 48.7% EM, substantially improving over GraphRAG Local Search but remaining below SAGE at $k = 20$. We also measure end-to-end latency (retrieval + answer generation) under sequential query execution to reflect a realistic deployment setting. SAGE averages 0.51 s per query, matching Community Expansion ($n = 1$) and outperforming GraphRAG Local Search (0.58 s), Community Expansion ($n = 3$) (0.68 s), and GraphRAG DRIFT (1.31 s). Together, these results show that SAGE delivers the strongest answer quality while maintaining low query latency.

Artifact Extracted	Tokens	SAGE	GraphRAG
Input Tokens	55127367	–	–
Entities	9677948	✓	✓
Relationships	9075366	×	✓
Facts	2123113	×	✓
Summaries	664953	✓	✓
Total Output Tokens		10342901	21541380

Table 4: Token usage for graph construction of OTT-QA.

Metadata Extraction Cost Analysis As shown in Table 4, both SAGE and GraphRAG produce outputs in the TOON format. However, GraphRAG requires entity extraction, relationship extraction, fact extraction, and summary generation to construct and index its graph, whereas SAGE only requires entities and summaries. By eliminating these, SAGE reduces output token consumption from 21.5M to 10.3M tokens, corresponding to a 52.0% reduction in graph construction cost. We provide the statistics for each baseline in Appendix C

Effect of metadata on graph building We analyze metadata similarities to guide graph construction. Strong correlations emerge within table-table pairs (title-description: 0.846, Table 19), document-document pairs (topic-content: 0.650, Table 20), and document-table pairs (0.517–0.691, Table 21), indicating that similarity in one metadata dimension often predicts similarity in others and helps form coherent graph neighborhoods.

Similarity scores follow a roughly bell-shaped distribution with a high-similarity tail, while entity overlap provides an important bridge between otherwise distinct topics. We therefore retain the top 5% of similarities to preserve meaningful connections without over-densifying the graph. Additional statistics are provided in Appendix E.

Structural effects of metadata signals Table 6 further reveals three cross-modal structural effects. **1) Entity-centric connectivity dominates but varies by modality:** tables form dense entity hubs, while text supports more semantically blended neighborhoods. **2) Modalities encode complementary signals:** text captures narrative structure, whereas tables provide schema-level alignment. **3) Cross-modal alignment emerges from signal composition:** document-table links

Category	Method	AMAZON		MAG		PRIME	
		Human $n = 81$	Synthetic $n = 1638$	Human $n = 84$	Synthetic $n = 2664$	Human $n = 98$	Synthetic $n = 2801$
Dense	ada-002	35.46	53.29	35.95	48.36	41.09	36.00
	multi-ada-002	40.22	55.12	39.85	50.80	47.21	38.05
	DPR	15.23	44.49	25.00	42.11	10.69	30.13
Sparse	BM25	15.23	53.77	32.46	45.69	42.32	31.25
Structural	QAGNN	21.54	52.05	28.76	46.97	17.62	29.63
Hybrid	4StepFocus	–	56.20	–	65.80	–	55.90
	FocusedR.	–	56.00	–	**78.80**	–	**65.50**
	Dense+BM25	24.30	51.49	46.64	51.49	52.65	39.11
Query Expansion	HyDe	39.25	53.71	37.23	50.02	47.70	43.55
	RAR	36.15	54.63	35.19	50.87	49.01	44.50
	AGR	37.27	53.38	37.23	51.89	49.65	46.63
	KAR	40.62	57.29	46.60	60.28	59.90	50.81
Agentic	ReACT	35.95	50.81	35.95	47.03	41.09	33.63
	Reflexion	35.95	54.70	35.95	49.55	41.09	38.52
Finetuned	AvaTaR	42.43	60.57	35.94	49.70	53.34	42.23
	mFAR	–	58.50	–	71.70	–	68.30
	MoR	–	59.90	–	75.00	–	63.50
SAGE(Ours)	SPARK	33.56	53.95	53.51	66.40	63.56	57.64
	SPARK +SAGE +Edge	36.93	54.12	55.90	68.60	65.85	58.42
	SPARK +SAGE-Edge	**42.05**	**58.28**	**56.40**	70.10	**66.37**	60.98

Table 5: Recall@20 performance on Human and Synthetic queries across all STaRK datasets. Highest non-finetuned method is bolded, second highest non-finetuned method is underlined. Note: FocusedR, 4StepFocus, mFAR, and MoR report results only on the Synthetic split; Human split results are not available from their publications or released artifacts.

Similarity metric	doc–doc	doc–tab	tab–tab
topic_similarity	0.350	0.247	0.263
content_similarity	0.389	0.350	0.404
column_similarity	0.000	0.242	0.409
entity_relationship_overlap	0.013	0.000	0.000
entity_count	0.682	0.477	0.849
topic_title_similarity	0.000	0.221	0.000
topic_summary_similarity	0.000	0.236	0.000
title_similarity	0.000	0.000	0.239
description_similarity	0.000	0.000	0.262

Table 6: Cross-modal similarity statistics across document and table pairs in OTT-QA.

arise from aggregating multiple weak but consistent signals rather than a single dominant metric. For STARK, an SKB, we do not recreate graph-building signals. Instead, we define edges between structured objects based on similarity in their descriptive metadata, while sparsely described entities are matched lexically via Cypher. Details provided in Appendix B.2.4.

3.2 STaRK: Semi-Structured Data Results

Effect of graph on agentic retriever Similar to our observations on OTT-QA, SAGE also improves Recall@20 for agentic retrievers across all STaRK subsets (Table 5). By expanding the retrieved set to include connected nodes, the graph reveals additional relevant nodes that the base agent misses, improving recall.

Are there dataset-specific patterns where graph augmentation helps more? Graph augmentation provides larger gains when nodes contain richer textual content. We compare two expansion policies: SPARK +SAGE-Edge skips neighbor expansion when Cypher queries specify explicit edges, while SPARK +SAGE +Edge always expands neighbors. On AMAZON, where nodes include detailed product metadata, SPARK +SAGE-Edge improves R@20 by +8.49 on Human queries and +4.33 on Synthetic queries. Datasets with sparser nodes, such as MAG and PRIME, show smaller gains of about 1-2 points. This suggests chunk-level graphs are most effective when nodes contain enough context to support multi-hop evidence chains that flat retrieval misses.

Method	#Retr.	#LLM	#Iter.	Set Join
4StepFocus	1 + n	2	n	Triplet cand.
FocusedR	≥3+n	3	$n_1 \times n_2$	Sym. cand
AvaTaR	n	n+1	n	–
KAR	e+3	2	**1**	–
ReACT	n	n+1	n	–
Reflexion	n	(n+1)E+R	nE+R	–
SAGE	**2**	**1**	**1**	–

Table 7: Theoretical runtime decomposition from method descriptions. #Retr. counts number of retrieval operations. n is a method-dependent number of reasoning steps. n_1, n_2 denote outer/inner iterations. e is the number of extracted entities. E and R denote episodes and calls. Set Join indicate candidate-neighborhood intersections.

How does this baseline+graph compare vs other state of the art methods SPARK +SAGE-Edge substantially outperforms traditional baselines, including Dense, Sparse, and Query Expansion methods, across all datasets (Table 5), while matching finetuned approaches such as AvaTaR, mFAR, and MoR. FocusedR performs better on MAG and

PRIME, where structured KG triples provide strong signals, but SPARK +SAGE-Edge achieves the best results on AMAZON by leveraging rich node content for chunk-level expansion. Importantly, it requires only a single LLM call and one retrieval step, making it more efficient and practical for large-scale deployment compared to baselines.

How efficient is our Agentic approach vs other state of the art methods Table 7 highlights key efficiency bottlenecks in existing retrieval methods, including repeated LLM calls, iterative retrieval loops, and symbolic set joins. Methods such as ReACT, Reflexion, and AvaTaR require multiple LLM invocations, while 4StepFocus, FocusedR, and KAR perform repeated retrieval or pruning steps. In contrast, SPARK uses a single LLM call, one retrieval, and one graph expansion, avoiding iterative loops and symbolic joins. This fixed execution path reduces latency and computational cost while maintaining competitive recall. Details are provided in Appendix F.

3.3 Ablation Study

Sensitivity Analysis: Effect of percentile during edge creation We study how percentile-based pruning shapes graph connectivity in OTT-QA by controlling the trade-off between connectivity and noise when adding similarity edges.

We compare two strategies: **OR logic**, which retains edges satisfying any similarity criterion to encourage connectivity, and **AND logic**, which requires all criteria and heavily prunes the graph.

Percentile	Logic	Total Edges	%Cand.	Avg Deg.	Density
80	OR	1,240,504	39.5%	109.2	0.0048
85	OR	969,911	30.9%	85.4	0.0038
90	OR	669,992	21.3%	59.0	0.0026
95	**OR**	**344,955**	**11.0%**	**30.4**	**0.0013**
97	OR	206,461	6.6%	18.2	0.0008
99	OR	60,371	1.9%	5.3	0.0002
80	AND	150,973	4.8%	13.3	0.0006
85	AND	101,963	3.2%	9.0	0.0004
90	AND	63,042	2.0%	5.6	0.0002
95	AND	30,920	1.0%	2.7	0.0001
97	AND	21,553	0.7%	1.9	0.0001
99	AND	13,434	0.4%	1.2	0.0001

Table 8: Impact of pruning percentile and logical edge selection on graph sparsity.

Table 8 shows that AND logic yields very low average degrees (1.2-2.7), fragmenting graphs and hindering multi-hop traversal. We therefore use OR logic, where the 95th percentile threshold balances coverage and sparsity: stricter thresholds fragment neighborhoods, while looser thresholds increase noise. This results in roughly 5-25 effective neighbors per chunk, preserving connectivity and keeping reranking tractable.

How much of the performance gain comes from the graph structure versus the graph expansion strategy? Both variants outperform the base SPARK model (Table 5), but SPARK +SAGE-Edge consistently achieves larger gains. Selectively skipping expansion for edge-aware queries avoids introducing irrelevant neighbors, while expansion remains useful when structural constraints are absent. This suggests that performance gains arise from both graph connectivity and query-aware expansion, which limits retrieval to semantically relevant neighbors.

What is the effect of the initial retriever on the graph-based expansion? Table 9 shows that the benefit of graph augmentation depends on the initial retriever. With the Hybrid baseline, lower Recall@20 leads to limited or sometimes negative gains from graph expansion. This occurs because the retriever ignores node types, while expansion favors embedding-similar nodes, often of the same type, which can introduce irrelevant additions.

	Human			Synthetic		
Method	A.	M.	P.	A.	M.	P.
Hybrid	24.3	46.6	52.7	51.5	51.5	39.1
Hybrid+Graph	31.1	46.4	53.4	53.8	52.9	38.4
Change (Δ)	+6.8	-0.2	+0.7	+2.3	+1.4	-0.7
SPARK	33.6	53.5	63.6	54.1	66.4	58.4
SPARK +SAGE	42.1	56.4	66.4	58.3	70.1	61.0
Change (Δ)	**+8.5**	**+2.9**	**+2.8**	**+4.2**	**+3.7**	**+2.6**

Table 9: Recall@20 before and after graph augmentation across different baseline retrievers on STaRK datasets. Hybrid refers to BM25+Cosine similarity. A., M., and P. stand for AMAZON, MAG, and PRIME, respectively.

In contrast, the agent retriever achieves higher Recall@20 and identifies node types. Graph expansion further improves performance across all datasets and query types, as stronger initial candidates and type-aware retrieval enable the addition of relevant nodes. Full Recall@k curves are provided in Appendix G.

3.4 Error Analysis on STaRK Human

We manually analyze retrieval failures on STaRK Human set and categorize errors into three classes:

- Data Errors reflect limitations in the dataset itself: These contain: **(1) Structural**, where there are missing or misdirected edges in the KG.**(2) Under-Specification**, where queries lack enough constraints to identify the correct entity. **(3) Evaluation Artifacts**, where gold annotations are incomplete, marking only a subset of valid answers.

- Cypher Generation Errors arise when translating natural language to graph queries: **(1) Constraint**, where numeric, temporal, or logical filters

are not correctly applied; and **(2) Structure**, where the system fails to select the correct node or edge types for graph traversal.

- Runtime Errors occur during query execution despite syntactically correct Cypher: **(1) Semantic**, where mismatches in units, terminology, or abbreviations prevent correct retrieval; and **(2) Over-Retrieval**, where queries return excessive candidates due to insufficient filtering.

Category	Method	Amazon	MAG	Prime
Data	Structural	5	6	4
	Under-Spec	4	5	3
	Eval Artifacts	7	4	4
Cypher	Constraint	36	30	31
	Structure	0	0	22
Runtime	Semantic	25	37	12
	Over-Retrieval	23	18	22
	Total	100	100	100

Table 10: Error distribution (%) across STaRK datasets.

Constraint extraction failures dominate on AMAZON and PRIME, highlighting challenges in mapping complex product attributes or biomedical predicates to precise graph queries. MAG errors mainly stem from semantic mismatches between technical abstracts, queries, and graph entities. Over-retrieval occurs across datasets due to broad filters, while PRIME also exhibits more graph structure errors from its richer schema. Data errors are minor and mostly appear in MAG because of incomplete gold annotations (Table 10).

Overall, the results highlight a trade-off between query translation, data quality, and semantic alignment: AMAZON errors arise from heterogeneous terminology and units, MAG from semantically complex content poorly captured by structured traversal, and PRIME from navigating its richer schema.

4 Related Work

Retrieval-augmented QA over heterogeneous data must capture textual relevance and structural information, including tables, linked documents, and relations. Key techniques are summarized below:

Table-centric retrieval and dataset discovery For table repositories, a major focus is retrieving *related* tables using structural criteria such as unionability and joinability. Systems like Starmie (Fan et al., 2022), DeepJoin (Dong et al., 2022), and WarpGate (Cong et al., 2022) learn column or table representations to support efficient join and union search, typically using ANN indexing. Other work explores specialized signals, such as sketch-based indexing for correlated dataset search (Santos et al., 2022). For end-to-end table QA, cascaded pipelines combining sparse filtering, dense retrieval, and reranking improve practical retrieval quality (Singh et al., 2025), while LLM-based dense retrieval methods like HyDE enable stronger zero-shot retrieval without labeled relevance data.

Graph-based indexing for retrieval and RAG Graph-based RAG methods organize information as graphs to retrieve connected context and aggregate evidence. Microsoft GraphRAG (Edge et al., 2024b) builds entity graphs with community summaries, while GRAG (Peng et al., 2024) retrieves textual subgraphs and incorporates graph topology into generation. PathRAG (Chen et al., 2025a) emphasizes redundancy pruning and path-based prompting. GNN-Ret (Li et al., 2024b) and ATLANTIC (Munikoti et al., 2023) use graph neural networks to encode passage- or document-level graph structure for retrieval. Knowledge-graph prompting (Wang et al., 2023) and Think-on-Graph (Sun et al., 2023) leverage graph traversal and iterative expansion for multi-hop reasoning. KET-RAG (Huang et al., 2025) combines a fixed entity-relation skeleton with a keyword-chunk graph, while KG^{2}RAG (Zhu et al., 2025) links chunks through extracted factual triples rather than broader semantic similarity. Surveys highlight the importance of graph structure and domain constraints in GraphRAG design (Han et al., 2024).

5 Conclusion

We present SAGE, a structure-aware graph retrieval framework that strikes a balance between the rigidity of KGs and the disjoint nature of flat chunking. Consistent with our contributions, SAGE augments flat retrieval with chunk-level graph expansion and is especially useful when building full KGs is impractical but structural signals remain implicit in metadata and chunk relationships.

On OTT-QA, similarity-driven expansion improves Recall@k by 2.8-5.7 points over hybrid baselines, retrieving more coherent multi-hop evidence. On STaRK, expansion approaches fine-tuned performance without training using Cypher-guided traversal to limit neighbor noise; overall, structure-aware retrieval leveraging graph connectivity can match or outperform strong baselines.

Future work includes instruction tuning the agent for more accurate Cypher generation and extending the framework to multimodal graphs that incorporate images through cross-modal links.

Limitations

Our study focuses on two representative regimes and uses LLM-based agents for STaRK, introducing higher latency and computational cost than traditional retrieval. This is a byproduct of enabling training-free symbolic Cypher generation, which improves schema-aware traversal and reduces neighbor noise in dense graphs. Performance depends on the initial seed retriever, as in all multi-stage retrieval systems, where downstream expansion operates on first-stage candidates; thus, retriever choice is inherently data- and modality-dependent rather than a limitation of the method. This motivates our SPARK agent, which serves as a more structure-aware initial retriever compared to the flat hybrid retriever used in our main setup.

SAGE performs best on nodes with rich text (e.g., abstracts, product descriptions, symptoms) but is less effective on graphs with mostly atomic entities, explaining smaller gains on STaRK MAG and PRIME. Error analysis shows constraint extraction and filtering (30-36%) as the main failure mode, reflecting the inherent difficulty of mapping natural language to structured predicates.

Despite this, SAGE improves recall over the agentic baseline, retrieves multi-hop evidence missed by flat retrieval, and achieves 2.8-5.7 point gains on OTT-QA while approaching fine-tuned baselines on STaRK without training.

Ethical Statement

We evaluate on publicly available benchmarks (OTT-QA and STaRK) released for research use under their respective licenses. Our framework operates on graph structures and does not collect any new user data or infer personal or demographic attributes.

Importantly, our agentic approach accesses only graph topology and schema information rather than actual node content during query planning, which provides an additional layer of privacy protection and reduces exposure to sensitive information. We aim to benefit the research community by enabling more efficient structure-aware retrieval across heterogeneous information sources.

As with any retrieval system, our approach could be misused in real deployments to surface or combine information without authorization. However, our work is intended solely for academic research and is not designed for deployment in surveillance, decision-making, or other high-stakes settings. Any practical use should follow standard data-governance practices (access control, auditing, and privacy safeguards) and undergo appropriate oversight. To support reproducibility, we document dataset splits, prompts, and decoding settings.

We used AI tools to assist with the writing process and improve the presentation.

References

2013. SPARQL 1.1 Query Language. Technical report, W3C.

Negar Arabzadeh, Xinyi Yan, and Charles LA Clarke. 2021. Predicting efficiency/effectiveness trade-offs for dense vs. sparse retrieval strategy selection. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 2862–2866.

Derian Boer, Fabian Koch, and Stefan Kramer. 2024. Harnessing the power of semi-structured knowledge and llms with triplet-based prefiltering for question answering. *arXiv preprint arXiv:2409.00861*.

Derian Boer, Stephen Roth, and Stefan Kramer. 2025. Focus, merge, rank: Improved question answering based on semi-structured knowledge bases. *arXiv preprint arXiv:2505.09246*.

Boyu Chen, Zirui Guo, Zidan Yang, Yuluo Chen, Junze Chen, Zhenghao Liu, Chuan Shi, and Cheng Yang. 2025a. Pathrag: Pruning graph-based retrieval augmented generation with relational paths. *arXiv preprint arXiv:2502.14902*.

Peter Baile Chen, Yi Zhang, Mike Cafarella, and Dan Roth. 2025b. Can we retrieve everything all at once? ARM: An alignment-oriented LLM-based retrieval method. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 30298–30317, Vienna, Austria. Association for Computational Linguistics.

Wenhu Chen, Ming-Wei Chang, Eva Schlinger, William Wang, and William W Cohen. 2020. Open question answering over tables and text. *arXiv preprint arXiv:2010.10439*.

Xilun Chen, Kushal Lakhotia, Barlas Oguz, Anchit Gupta, Patrick Lewis, Stan Peshterliev, Yashar Mehdad, Sonal Gupta, and Wen-tau Yih. 2022. Salient phrase aware dense retrieval: can a dense retriever imitate a sparse one? In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 250–262.

Xinran Chen, Xuanang Chen, Ben He, Tengfei Wen, and Le Sun. 2024. Analyze, generate and refine: Query expansion with LLMs for zero-shot open-domain QA. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11908–11922, Bangkok, Thailand. Association for Computational Linguistics.

Tianji Cong, James Gale, Jason Frantz, HV Jagadish, and Çağatay Demiralp. 2022. Warpgate: A semantic join discovery system for cloud data warehouses. *arXiv preprint arXiv:2212.14155*.

Yuyang Dong, Chuan Xiao, Takuma Nozawa, Masafumi Enomoto, and Masafumi Oyamada. 2022. Deepjoin: Joinable table discovery with pre-trained language models. *arXiv preprint arXiv:2212.07588*.

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

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

Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée Miller. 2022. Semantics-aware dataset discovery from data lakes with contextualized column-based representation learning. *arXiv preprint arXiv:2210.01922*.

Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. 2018. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 international conference on management of data*, pages 1433–1445.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2023. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1762–1777.

Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A Rossi, Subhabrata Mukherjee, Xianfeng Tang, et al. 2024. Retrieval-augmented generation with graphs (graphrag). *arXiv preprint arXiv:2501.00309*.

Junjie Huang, Wanjun Zhong, Qian Liu, Ming Gong, Daxin Jiang, and Nan Duan. 2022. Mixed-modality representation learning and pre-training for joint table-and-text retrieval in OpenQA. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 4117–4129, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yiqian Huang, Shiqi Zhang, and Xiaokui Xiao. 2025. Ket-rag: A cost-efficient multi-granular indexing framework for graph-rag. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25, page 1003âĂŞ1012, New York, NY, USA. Association for Computing Machinery.

Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in Neural Information Processing Systems*, 37:59532–59569.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick SH Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *EMNLP (1)*, pages 6769–6781.

Yongjia Lei, Haoyu Han, Ryan A Rossi, Franck Dernoncourt, Nedim Lipka, Mahantesh M Halappanavar, Jiliang Tang, and Yu Wang. 2025. Mixture of structural-and-textual retrieval over text-rich graph knowledge bases. *arXiv preprint arXiv:2502.20317*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Yihao Li, Ru Zhang, and Jianyi Liu. 2024a. An enhanced prompt-based llm reasoning scheme via knowledge graph-integrated collaboration. In *International Conference on Artificial Neural Networks*, pages 251–265.

Zijian Li, Qingyan Guo, Jiawei Shao, Lei Song, Jiang Bian, Jun Zhang, and Rui Wang. 2024b. Graph neural network enhanced retrieval for question answering of llms.

Kaixin Ma, Hao Cheng, Xiaodong Liu, Eric Nyberg, and Jianfeng Gao. 2022. Open-domain question answering via chain of reasoning over heterogeneous knowledge. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 5360–5374, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Kaixin Ma, Hao Cheng, Yu Zhang, Xiaodong Liu, Eric Nyberg, and Jianfeng Gao. 2023. Chain-of-skills: A configurable model for open-domain question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1599–1618, Toronto, Canada. Association for Computational Linguistics.

Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836.

Priyanka Mandikal and Raymond Mooney. 2024. Sparse meets dense: A hybrid approach to enhance scientific document retrieval. *arXiv preprint arXiv:2401.04055*.

Costas Mavromatis and George Karypis. 2024. Gnn-rag: Graph neural retrieval for large language model reasoning. *arXiv preprint arXiv:2405.20139*.

Sai Munikoti, Anurag Acharya, Sridevi Wagle, and Sameera Horawalavithana. 2023. Atlantic: Structure-aware retrieval-augmented language model for interdisciplinary science.

Neo4j, Inc. 2026. *Neo4j Graph Database*.

OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, and et al. 2024. Gpt-4o system card.

Eunhwan Park, Sung min Lee, Dearyong Seo, Seonhoon Kim, Inho Kang, and Seung-Hoon Na. 2023. Rink: Reader-inherited evidence reranker for table-and-text open domain question answering. In *AAAI Conference on Artificial Intelligence*.

Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. 2024. Graph retrieval-augmented generation: A survey. *ACM Transactions on Information Systems*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

AÃl'cio Santos, Aline Bessa, Christopher Musco, and Juliana Freire. 2022. A sketch-based index for correlated dataset search. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 2928–2941.

Tao Shen, Guodong Long, Xiubo Geng, Chongyang Tao, Yibin Lei, Tianyi Zhou, Michael Blumenstein, and Daxin Jiang. 2024. Retrieval-augmented retrieval: Large language models are strong zero-shot retriever. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 15933–15946, Bangkok, Thailand. Association for Computational Linguistics.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Adarsh Singh, Kushal Raj Bhandari, Jianxi Gao, Soham Dan, and Vivek Gupta. 2025. Craft: Training-free cascaded retrieval for tabular qa. *arXiv preprint arXiv:2505.14984*.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Heung-Yeung Shum, and Jian Guo. 2023. Think-on-graph: Deep and responsible reasoning of large language model with knowledge graph.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 33:5776–5788.

Yu Wang, Nedim Lipka, Ryan A. Rossi, Alexa Siu, Ruiyi Zhang, and Tyler Derr. 2023. Knowledge graph prompting for multi-document question answering.

Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis Ioannidis, Karthik Subbian, Jure Leskovec, and James Y Zou. 2024a. Avatar: Optimizing llm agents for tool usage via contrastive reasoning. *Advances in Neural Information Processing Systems*, 37:25981–26010.

Shirley Wu, Shiyu Zhao, Michihiro Yasunaga, Kexin Huang, Kaidi Cao, Qian Huang, Vassilis N. Ioannidis, Karthik Subbian, James Zou, and Jure Leskovec. 2024b. Stark: Benchmarking llm retrieval on textual and relational knowledge bases. In *NeurIPS Datasets and Benchmarks Track*.

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*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. 2021. Qa-gnn: Reasoning with language models and knowledge graphs for question answering. *arXiv preprint arXiv:2104.06378*.

Haoyu Zhang, Jun Liu, Zhenhua Zhu, Shulin Zeng, Maojia Sheng, Tao Yang, Guohao Dai, and Yu Wang. 2024. Efficient and effective retrieval of dense-sparse hybrid vectors using graph-based approximate nearest neighbor search. *arXiv preprint arXiv:2410.20381*.

Wanjun Zhong, Junjie Huang, Qian Liu, Ming Zhou, Jiahai Wang, Jian Yin, and Nan Duan. 2022. Reasoning over hybrid chain for table-and-text open domain qa. *arXiv preprint arXiv:2201.05880*.

Xiangrong Zhu, Yuexiang Xie, Yi Liu, Yaliang Li, and Wei Hu. 2025. Knowledge graph-guided retrieval augmented generation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 8912–8924, Albuquerque, New Mexico. Association for Computational Linguistics.

A Example of Document Chunking

Consider a document consisting of sentences $\{s_i\}_{i=1}^{6}$:

- s_1: The company reports strong quarterly earnings driven by enterprise software adoption and steady cloud revenue growth.
- s_2: Operating margins improve due to cost optimization, improved infrastructure efficiency, and reduced customer acquisition costs.
- s_3: Revenue growth remains robust across enterprise and cloud segments, supported by continued customer expansion.
- s_4: Looking ahead, the company outlines a more cautious outlook due to macroeconomic uncertainty and potential demand slowdowns in key markets.
- s_5: Management emphasizes a shift in strategy toward higher-margin enterprise offerings and long-term subscription stability.
- s_6: The firm expects sustained performance improvements driven by disciplined investment and recurring revenue growth.

We construct overlapping context windows $w_i = [s_{i-1}, s_i, s_{i+1}]$ with boundary truncation and encode each window using a sentence encoder (Reimers and Gurevych, 2019), yielding embeddings $\mathbf{h}_i$.

Semantic continuity is measured via cosine similarity:

$$s_i = \cos(\mathbf{h}_i, \mathbf{h}_{i+1}).$$

Let $\{s_i\} = \{0.92, 0.89, 0.41, 0.87, 0.90\}$ denote adjacent-window similarities. We define a threshold:

$$\theta = \text{Percentile}_{20}(\{s_i\}),$$

and introduce a boundary at positions where $s_i \leq \theta$.

In this example, the sharp drop at $s_3 = 0.41$ reflects a transition from *current financial performance* (earnings and growth) to *forward-looking strategy and outlook*, yielding two segments: *(i)* s_1–s_3 (performance reporting) and *(ii)* s_4–s_6 (outlook and strategy).

Intuition. Unlike fixed-length segmentation, boundaries are induced by embedding-space coherence, ensuring that chunks align with semantic transitions rather than sentence count. In the case where there is no drop, the entire document is treated as semantically coherent. In this case, we split the document by sentences, keeping a minimum of 4 and a maximum of 6 sentences.

B Dataset Statistics

B.1 OTT-QA

OTT-QA consists of a heterogeneous graph connecting document and table chunks (Table 11). While table chunks are relatively few, they are densely linked to documents, enabling cross-modal retrieval between structured and unstructured sources. The large number of document-document edges reflects strong topical overlap across documents, which is particularly useful for graph-based context propagation.

Component	**Count**
Document chunks	21,912
Table chunks	974
Total Nodes	**22,886**
Table–Table edges	66,003
Document–Table edges	162,374
Document–Document edges	523,459
Total Edges	**751,836**

Table 11: OTT-QA dataset statistics showing node types, edge types, and their counts.

B.2 STaRK

B.2.1 AMAZON

The AMAZON dataset originally contains four node types: `review`, `brand`, `color`, and `category`. The latter three (`brand`, `color`, `category`) can alternatively be represented as metadata attributes rather than separate nodes. We observe that all product nodes contain a `reviews` key that stores tabular review data.

For the dense+sparse (hybrid) retrieval approach, we treat this reviews data as a table for retrieval and expansion (Table 12). Since Neo4j is a graph database and does not natively support tabular structures like SQL databases, we convert each review into a separate node connected to its corresponding product node (Table 13).. This graph-native representation enables efficient querying in cases where

we need to aggregate data (e.g., star ratings) across multiple reviews for the same product.

Node Type	Count
Review	957,192
Product	957,192
Total	**1,914,384**

Table 12: AMAZON node type distribution from STaRK dataset for Hybrid retrieval.

Node Type	Count
Review	12,963,012
Product	957,192
Total	**13,920,204**

Table 13: AMAZON node type distribution from STaRK dataset for Neo4j queries.

Edge Type	Count
HAS_REVIEW	12,963,012
ALSO_BUY	1,663,562
ALSO_VIEW	1,362,680
IS_SIMILAR	24,886,992
Total	**40,876,246**

Table 14: AMAZON edge type distribution from STaRK dataset.

B.2.2 MAG

MAG represents a large scholarly graph with diverse entity types but comparatively limited textual richness per node (Tables 15 and 16). While citation and authorship edges provide strong structural signals, the effectiveness of semantic graph expansion is constrained by shorter textual descriptions.

Node Type	Count
Author	1,104,554
Paper	700,244
Field	59,469
Institution	8,701
Total	**1,872,968**

Table 15: MAG node type distribution from STaRK dataset.

Edge Type	Count
CITES	9,719,488
HAS_FIELD	7,243,269
AUTHORED	6,768,883
ASSOCIATED_WITH	167,482
IS_SIMILAR	12,954,514
Total	**36,853,636**

Table 16: MAG edge type distribution from STaRK dataset.

B.2.3 PRIME

PRIME is a highly structured biomedical graph with many fine-grained entity and relation types (Tables 17 and 18). Although the graph is rich in relational semantics, node-level textual content is often sparse, making effective graph expansion more sensitive to the quality of initial retrieval.

Node Type	Count
BiologicalProcess	28,642
Gene	27,671
Disease	17,080
Phenotype	15,311
Anatomy	14,035
MolecularFunction	11,169
Drug	7,957
CellularComponent	4,176
Pathway	2,516
Exposure	818
Total	**129,375**

Table 17: PRIME node type distribution from STaRK dataset.

Edge Type	Count
EXPRESSED_IN	3,036,406
SYNERGISTIC_WITH	2,672,628
AFFILIATED_WITH	1,029,162
INTERACTS_WITH	686,550
INTERACTS_WITH_PROTEIN	642,150
PHENOTYPE_PRESENT	300,634
PARENT_OF	281,744
HAS_SIDE_EFFECT	129,568
CONTRAINDICATED_FOR	61,350
NOT_EXPRESSED_IN	39,774
TARGETS	32,760
INDICATED_FOR	18,776
INHIBITS_ENZYME	10,634
TRANSPORTED_BY	6,184
OFF_LABEL_USE	5,136
LINKED_TO	4,608
PHENOTYPE_ABSENT	2,386
ACTS_AS_CARRIER	1,728
IS_SIMILAR	963,104
Total	**9,825,282**

Table 18: PRIME edge type distribution from STaRK dataset.

B.2.4 Graph Building Signals

Since each node type contains multiple metadata fields, we use only fields with descriptive textual content and exclude identifiers such as names, dates, and IDs. For dataset-specific graphs, Amazon uses product names, descriptions, and product details; MAG uses paper titles, abstracts, and fields of study; and PRIME uses descriptions of drugs, diseases, genes/proteins, and pathways. Semi-structured chunk edges are constructed by linking objects based on semantic similarity between their metadata representations.

C OTT-QA Baselines

Prior work mainly targets table-level retrieval, such as *CRAFT* (Singh et al., 2025) and *OTTeR* (Huang et al., 2022), rather than chunk-level retrieval. Many methods rely on predefined table-document links, including *COS* (Ma et al., 2023), *RINK* (Park et al., 2023), *CARP* (Zhong et al., 2022), and *Dual Reader-Parser*. Methods that ignore these links use different setups: *ARM* (Chen et al., 2025b) adopts gold-set evaluation, while *CORE* (Ma et al., 2022) attempts to reconstruct the links, preventing direct comparison.

C.1 GraphRAG

Building (Edge et al., 2024b) GraphRAG builds an entity-centric knowledge graph by extracting entities and relationships from document chunks using an LLM. The resulting graph is partitioned into communities using graph clustering, and each community is summarized into a textual report. This preprocessing stage produces entity embeddings, graph structure, and community summaries that are later used during retrieval.

C.2 GraphRAG

- Number of graph nodes: 135,785
- Number of graph edges: 3,670,174

Querying: GraphRAG Local Search. At query time, Local Search embeds the user query and retrieves the most relevant entities from the knowledge graph. These entities act as anchors for extracting their immediate graph neighborhoods, including neighboring entities, relationships, associated text units, and local community reports. The retrieved context is ranked by relevance and assembled into the prompt for answer generation. As a result, Local Search functions as a focused retrieval strategy optimized for entity-centric questions requiring localized graph context.

Querying: GraphRAG DRIFT Search. DRIFT Search extends Local Search through iterative graph exploration. Rather than relying on a single retrieval pass, it uses the LLM to generate intermediate hypotheses and follow-up questions that guide subsequent retrieval steps. This process progressively expands the search frontier, incorporating both deeper entity-level neighborhoods and broader community-level information. The final context is therefore constructed dynamically based on the retrieval trajectory, enabling adaptive exploration beyond the immediate graph neighborhood. To bound retrieval cost and avoid degenerate search loops, we limit DRIFT to two expansion iterations.

C.3 Community Expansion

Index Construction We build an offline chunk graph where each document/table chunk is a node and edges are added from metadata-based semantic similarities. Community assignments are then obtained with the Leiden algorithm on this weighted graph (the number of communities is not fixed and emerges from graph structure), producing an inverted index from community IDs to member chunks.

Edges are retained using only a 95th-percentile similarity cutoff, without additional multi-signal pruning or extra filtering rules, and Leiden is applied directly on this graph.

- Number of communities: 2,028
- Minimum community size: 2
- Maximum community size: 351
- Average community size: 11.20
- Median community size: 7.00
- Population standard deviation: 17.48

Community size distribution

- 1–10: 1,408
- 11–25: 458
- 26–50: 116
- 51–100: 35
- 101–250: 9
- 251+: 2

Top 10 largest communities

- community_0: 351
- community_1: 268
- community_16: 202
- community_12: 190
- community_2: 170
- community_3: 132
- community_9: 130
- community_5: 122
- community_252: 121
- community_4: 109

Querying At query time, we retrieve top-k chunks using a hybrid BM25+dense search over the full corpus. For $n = 1$, we identify the communities containing these top-k chunks and expand the candidate pool to all chunks within those communities, followed by re-ranking with the same hybrid scorer. For $n = 3$, we additionally use embedding similarity to select the two closest communities to the original set, and perform retrieval over the union of these communities, forming the final expanded candidate pool for ranking.

C.4 Comparison vs SAGE

D STaRK Baselines

ada-002 This dense retriever embeds each query and node independently using the text-embedding-ada-002 model, then ranks candidates by embedding similarity. It provides a strong semantic retrieval baseline for nodes with descriptive text. However, it does not use relation types, graph topology, or schema constraints during retrieval.

multi-ada-002 This baseline uses a multilingual dense embedding model in the same bi-encoder retrieval setup as ada-002. It is intended to improve robustness to lexical variation and semantic paraphrase across diverse node descriptions. Like other dense retrievers, it remains structure-agnostic and cannot explicitly follow graph relations.

DPR (Karpukhin et al., 2020) DPR is a supervised dense passage retriever originally designed for open-domain question answering. It encodes queries and candidates separately and ranks them by vector similarity. On STaRK, it serves as a standard neural retrieval baseline but does not exploit typed edges or schema information.

BM25 BM25 is a sparse lexical retriever that ranks nodes using exact and partial term overlap between the query and node text. It is a strong baseline when entity names or attributes appear verbatim in the corpus. Its main limitation is that it cannot capture paraphrases or multi-hop structural dependencies.

BM25+Dense This hybrid baseline combines sparse lexical matching from BM25 with dense semantic similarity from embedding-based retrieval. The goal is to improve coverage by balancing exact term matches with semantic matches. Despite this stronger initialization, it still treats nodes independently and ignores schema-valid paths during retrieval.

4StepFocus (Boer et al., 2024) 4StepFocus is a hybrid SKB retriever that first extracts structured constraints, prunes the candidate space symbolically, then applies vector similarity scoring and reranking. It is designed to reduce search space before semantic ranking. This makes it stronger than flat retrieval, but it relies on iterative candidate filtering and more complex execution.

FocusedR (Boer et al., 2025) FocusedR generates structured graph queries, grounds symbolic constraints, expands candidate sets adaptively, and finally reranks hybrid candidates. It is a strong retrieval baseline for explicit KGs because it uses both graph structure and unstructured text. Its main drawback is higher complexity due to repeated symbolic pruning and candidate expansion.

QA-GNN (Yasunaga et al., 2021) QA-GNN is a graph-neural baseline that scores candidates by combining node features with graph connectivity in a learned message-passing framework. It represents a structural retrieval baseline that directly encodes topology. Unlike our method, it depends on learned graph reasoning rather than retriever-agnostic expansion over an offline graph.

HyDE (Gao et al., 2023) HyDE is a query expansion method that first asks an LLM to generate a hypothetical answer document, then retrieves

nodes using embeddings of that generated text. The generated pseudo-document helps bridge the gap between short queries and longer candidate descriptions. However, the method remains fundamentally flat and does not use graph structure during retrieval.

AGR (Chen et al., 2024) AGR is an LLM-based query expansion baseline that enriches the original query with additional context intended to improve downstream retrieval. It aims to recover missing semantic cues before ranking candidates. Like other expansion-only methods, it improves query representation but does not explicitly traverse graph structure.

RAR (Shen et al., 2024) RAR is another retrieval-oriented query rewriting baseline that reformulates the input question to produce a retrieval-friendly query. Its strength lies in better aligning the query with candidate text through reformulation. However, it still operates over flat retrieval and cannot enforce typed multi-hop paths.

KAR (Xia et al., 2024) KAR is a knowledge-aware query expansion method that extracts entities, retrieves their neighbors, constructs supporting triples, and then generates an expanded query for final retrieval. It injects structured context into the query before retrieval rather than expanding over a pre-built graph at inference time. This makes it more structured than flat baselines, but it still requires multiple retrieval operations and LLM calls.

ReACT (Yao et al., 2022) ReACT is an agentic baseline that interleaves reasoning and retrieval in a think-act-observe loop. At each step, the model decides what retrieval action to take next based on prior observations. This enables flexible multi-step exploration, but incurs higher latency because retrieval is embedded inside an iterative LLM loop.

Reflexion (Shinn et al., 2023) Reflexion extends ReACT by adding self-reflection after failed trajectories, allowing the agent to critique prior mistakes and retry with updated reasoning. This can improve difficult retrieval problems by re-planning after failure. The tradeoff is even higher inference cost due to repeated episodes and extra LLM calls.

mFAR (Li et al., 2024a) mFAR is a fine-tuned semi-structured retriever trained specifically for retrieval over structured or semi-structured corpora. It benefits from supervised optimization tailored to the target retrieval task, giving it an advantage over zero-shot baselines. In our comparisons, it serves as a strong learned upper baseline rather than a training-free method.

MoR (Lei et al., 2025) MoR is another fine-tuned retrieval model for semi-structured knowledge bases that learns task-specific ranking behavior from supervision. It is designed to better capture relation-aware retrieval patterns than generic dense or sparse baselines. Like mFAR, it is not directly comparable to training-free methods on efficiency, but provides a useful performance reference.

AvaTaR (Wu et al., 2024a) AvaTaR is a fine-tuned agentic retriever that improves tool-using retrieval through trajectory optimization and contrastive supervision. Instead of relying only on zero-shot prompting, it learns better action policies from successful and failed retrieval traces. This makes it a strong learned agent baseline, especially on structured retrieval tasks.

E OTTQA graph building statistics

E.1 Quality of Metadata Extracted

The edges are created from metadata generated by an LLM, which introduces a dependence on generation quality. If the model produces generic titles or topics, the resulting graph can become overly dense or connect irrelevant chunks.

We therefore do not deploy LLM-generated metadata without checking it first. Instead, we design prompts and test them on a small sample of chunks to inspect the resulting titles, topics, and entities. If the outputs appear too generic or unhelpful, we revise the prompts and repeat the sample-based check until the metadata is clear and discriminative. Only after this validation step do we run the prompts at scale for graph construction. This iterative sampling process reduces the risk of low-quality generations before the full pipeline is executed.

Sections E.2, E.3, and E.4 provide empirical evidence that the metadata does not collapse into generic patterns. Title and description similarity distributions are bell-shaped with a moderate center around 0.3, far below the 95th-percentile edge threshold of 0.63. If titles were generic, these similarities would cluster in a narrow high-similarity band; instead, the observed spread indicates differentiated, chunk-specific generations.

We further show that LLM-generated metadata aligns with the underlying content. Scatter plots and correlation analyses in Sections E.2, E.3, and E.4 demonstrate strong relationships between titles, descriptions, topics, and source text, including a topic-content correlation of 0.650. These diagonal trends would not appear if the metadata were generic; instead, they indicate that the metadata tracks content rather than reflecting templated labeling. The metadata extraction prompts themselves are provided in Section K.

Finally, 95th-percentile pruning acts as a safeguard. Even if some metadata were generic, it would affect only a small high-similarity tail. The retained graph remains controlled, with average degree around 30 and only about 11% of candidate edges preserved, clearly separated from denser settings. This cutoff limits connectivity and prevents generic metadata from inflating the graph.

E.2 Table-Table Similarity Analysis

We analyze similarity statistics between table chunks in OTT-QA to evaluate whether LLMs are necessary for graph construction.

Column similarity and entity distribution Figure 4 show the distribution of column similarity and number of entities per chunk. Column similarity is roughly bell-shaped (normal distribution), while entity counts are strongly left-skewed - most chunks contain only one entity. These figures highlight that high semantic similarity exists even for single-entity chunks.

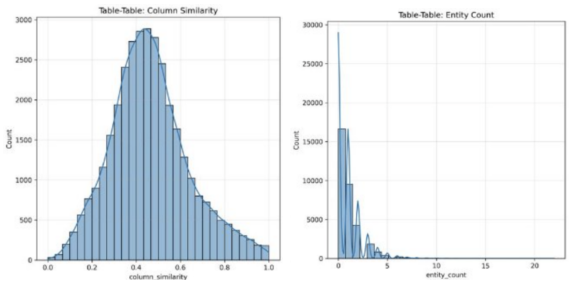

Figure 4: Left: column similarity distribution. Right: number of entities per table chunk (left-skewed).

Description and title similarity Figure 5 shows the distributions of description and title similarity. Both are bell-shaped with a slight high-end tail, indicating that some tables are highly similar in title or description.

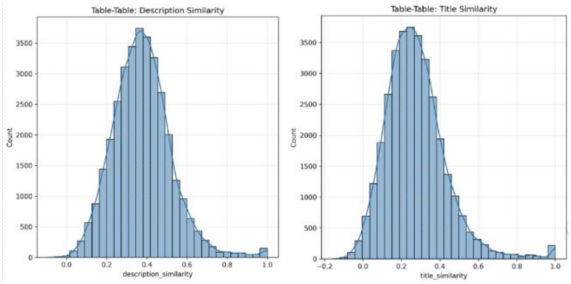

Figure 5: Left: description similarity. Right: title similarity. Both distributions are similar, with a slight rise at high similarity.

Column-title, column-description, title-description scatter plots Figure 6 shows scatter plots of column-title, column-description, and title-description similarities, respectively. Strong diagonal trends indicate that similar columns tend to co-occur with similar titles and descriptions.

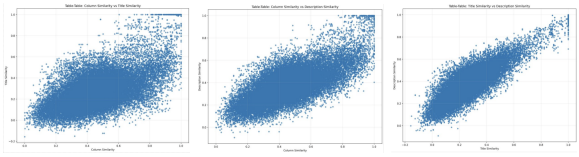

Figure 6: Scatter plots of column-title (left), column-description (center), and title-description (right) similarities. Diagonal trends indicate strong alignment.

Correlation matrix Table 19 shows the correlation matrix of all table-level similarity metrics. Column, title, and description similarities are strongly correlated, with title and description similarity correlation 0.846.

	(1)	(2)	(3)	(4)
(1) Col Sim	1.000	0.165	0.577	0.696
(2) Ent Cnt	0.165	1.000	0.179	0.189
(3) Titl Sim	0.577	0.179	1.000	0.846
(4) Desc Sim	0.696	0.189	0.846	1.000

Table 19: Table-Table Correlation Matrix. Abbreviations: (1) column_similarity, (2) entity_count, (3) title_similarity, and (4) description_similarity.

E.3 Document-Document Similarity Analysis

We analyze similarity statistics between document chunks in OTT-QA to assess semantic content and the need for LLM-based processing.

Entity and event counts per chunk Figure 7 shows the number of entities and events per document chunk. Both distributions are strongly left-skewed, with most chunks containing only a single entity or event. This indicates that document

chunks are small and focused, and additional LLM-based extraction or summarization would provide minimal benefit.

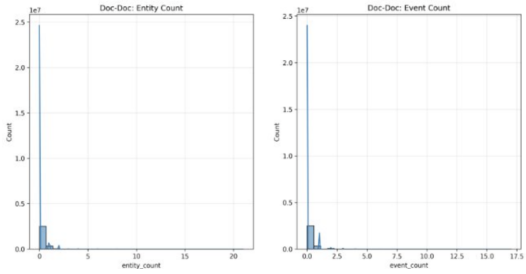

Figure 7: Distribution of entities and events per document chunk. Most chunks contain only one entity, reducing the need for LLM-based extraction.

Topic and content similarity distributions Figure 8 shows histograms of topic similarity and content similarity between document chunks. Both distributions are roughly bell-shaped with a small spike near 1, indicating that while most document pairs are moderately similar, a few are nearly identical.

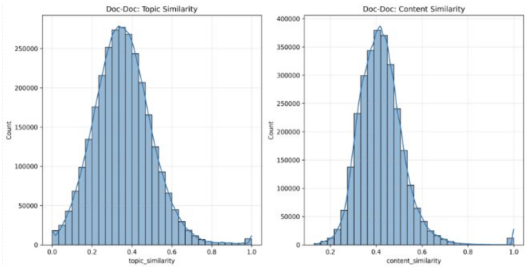

Figure 8: Histograms of topic similarity (left) and content similarity (right) between document chunks. Bell-shaped distributions with a high-similarity spike are observed.

Topic-content scatter plot Figure 9 shows a scatter plot of content similarity versus topic similarity for each document pair. Most points lie along the diagonal, confirming strong alignment between content and topic similarities.

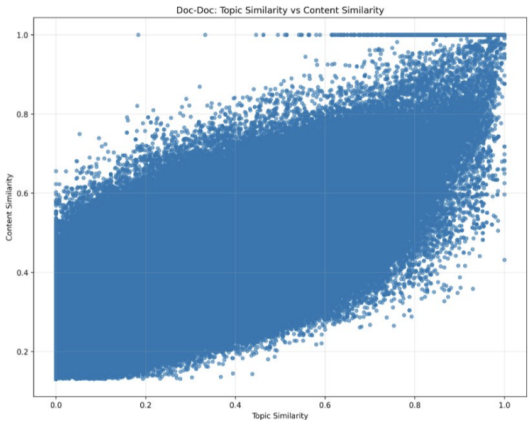

Figure 9: Scatter plot of content similarity versus topic similarity. Diagonal concentration indicates strong alignment between the two metrics.

Correlation matrix Table 20 shows the correlation matrix of document-document similarity metrics. Topic and content similarity are strongly correlated, while entity and event features provide complementary signals.

	(1)	(2)	(3)	(4)	(5)
(1) Top Sim	1.000	0.650	0.250	0.253	0.153
(2) Cont Sim	0.650	1.000	0.357	0.339	0.251
(3) Ent Rel	0.250	0.357	1.000	0.530	0.230
(4) Ent Cnt	0.253	0.339	0.530	1.000	0.235
(5) Evt Cnt	0.153	0.251	0.230	0.235	1.000

Table 20: Doc-Doc Correlation Matrix. Abbreviations: (1) topic_similarity, (2) content_similarity, (3) entity_relationship_overlap, (4) entity_count, and (5) event_count.

E.4 Document-Table Similarity Analysis

We analyze the semantic alignment between document and table chunks in OTT-QA to evaluate whether LLM-based processing is necessary.

Topic-title and topic-summary similarity Figure 10 shows the distributions of topic-title similarity and topic-summary similarity, respectively. Both curves are bell-shaped, indicating most document-table pairs are moderately similar, with few extreme cases. These two histograms are placed side by side for direct comparison.

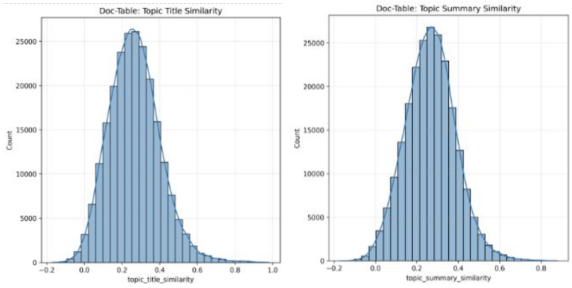

Figure 10: Left: topic-title similarity distribution. Right: topic-summary similarity distribution. Both are bell-shaped.

Column similarity and entity counts Figure 11 shows column similarity and number of entities per table chunk. Column similarity exhibits a bell-shaped distribution, while the entity count is left-skewed, consistent with document-document observations. Placing these side by side highlights the distributional patterns across structural and semantic features.

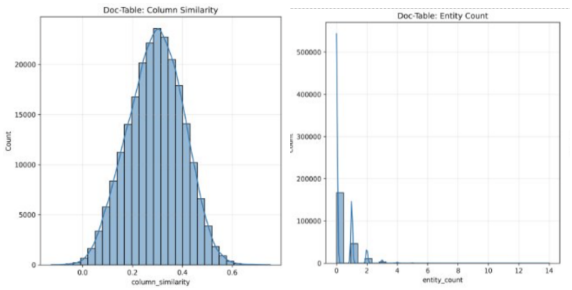

Figure 11: Left: column similarity distribution. Right: entity count per table chunk (left-skewed).

Scatter plots of topic, title, summary, and column similarities Figure 12 shows scatter plots for combinations of topic, title, summary, and column similarities. Diagonal trends indicate strong alignment between document and table chunks across multiple semantic dimensions.

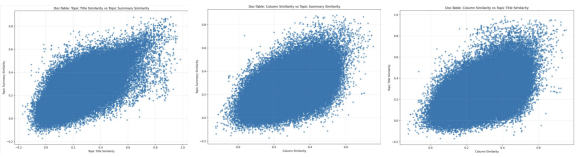

Figure 12: Scatter plots showing alignment between document and table chunks: topic-title vs topic-summary (left), column vs topic-summary (center), column vs topic-title (right). Diagonal trends indicate strong alignment.

Correlation matrix Table 21 shows the correlation matrix of document-table similarity metrics. Column similarity, title similarity, and summary similarity are moderately correlated, confirming that cross-modal semantic alignment can be captured using simple embeddings without LLM-based processing.

	(1)	**(2)**	**(3)**	**(4)**
(1) Col Sim	1.000	0.078	0.517	0.535
(2) Ent Count	0.078	1.000	0.107	0.107
(3) Title Sim	0.517	0.107	1.000	0.691
(4) Summ Sim	0.535	0.107	0.691	1.000

Table 21: Doc-Table Correlation Matrix. Abbreviations: (1) column_similarity, (2) entity_count, (3) topic_title_similarity, and (4) topic_summary_similarity

E.5 Communities

Community-based partitioning introduces its own set of limitations. If clustering methods such as k-means are used, selecting an appropriate value of k is non-trivial and can significantly affect downstream performance. Without careful tuning, clustering may produce highly imbalanced partitions, as reflected in the observed distribution, where a small number of communities become disproportionately large.

This imbalance can lead to inefficiencies during retrieval: if a large fraction of the corpus collapses into a few dominant communities, the system must still search through extensive candidate sets during inference. Conversely, if semantically related content is split across multiple communities, relevant evidence may be fragmented, requiring broader exploration across many partitions. In extreme cases, corpora that are topically concentrated may collapse into a single large community, while heterogeneous corpora may produce overly dispersed structures that degrade retrieval efficiency.

E.6 SAGE

SAGE addresses several of the limitations observed in GraphRAG-style constructions. By maintaining fixed chunks as nodes, it avoids the instability associated with dynamically constructed or excessively dense edge structures. Instead of relying on potentially noisy or semantically redundant edges, connectivity is defined based on whether chunks exhibit meaningful semantic similarity, improving grounding quality.

Unlike community-based methods, SAGE mitigates imbalance issues by construction, as retrieval operates over a percentile-based or uniformly constrained structure rather than emergent cluster sizes. The retrieval process can be viewed as a two-stage mechanism: an initial global retrieval step identifies a coarse set of relevant regions in the corpus,

followed by graph expansion and pruning that functions as a localized search, concentrating computation on the most relevant evidence. This design improves both retrieval efficiency and robustness while maintaining coverage across the underlying knowledge space.

F Runtime Analysis of all State of the Art baselines for STaRK

F.1 4StepFocus

4StepFocus processes queries over semi-structured knowledge bases (SKBs) through symbolic prefiltering, followed by vector similarity search and LLM-based reranking.

Pipeline:

1. **LLM Extraction:** The LLM extracts triplets T and the target variable x_{target} from the query q.
2. **Symbolic Prefiltering (SUBSTITUTE):** Iteratively intersects candidate sets with KG neighbors until convergence, producing a filtered candidate set C_{filtered}.
3. **Vector Similarity Scoring (VSS):** Scores C_{filtered} using unstructured data, returning the top-$k_{\max}$ candidates along with additional relevant candidates.
4. **LLM Reranking:** Reranks candidates using the SKB context.

SUBSTITUTE Algorithm:
Runtime Analysis:

- **Step 1:** $O(1)$ - single LLM call.
- **Step 2:** $O(n \cdot |T| \cdot \deg)$ - n iterations over $|T|$ triplets with average node degree $\deg$.
- **Step 3:** $O(|C_{\text{filtered}}|)$ - VSS applied to filtered candidates.
- **Step 4:** $O(1)$ - single LLM call for reranking.

Total Complexity:

$$O(n \cdot |T| \cdot \deg + |C_{\text{filtered}}|)$$

F.2 FocusedR

FocusedR combines Cypher query generation, symbolic grounding, and hybrid retrieval (Cypher-constrained plus fallback Vector similarity search(VSS)) before LLM-based reranking.

Pipeline:

Algorithm 1: SUBSTITUTE: Symbolic Prefiltering for Candidate Selection

Input: Candidate set C, triplets $T = \{(h_i, e_i, t_i)\}$, target variable x_{target}
Output: Filtered candidate set C_{filtered}
foreach *variable* x **do**
  substitute$[x] \leftarrow \{c \in C \mid \text{type}(c) = \text{type}(x)\}$
end
repeat
  foreach $(h_i, e_i, t_i) \in T$ **do**
    substitute$[h_i] \leftarrow$ substitute$[h_i] \cap$neighbors$(h_i, e_i, -)$
    substitute$[t_i] \leftarrow$ substitute$[t_i] \cap$ neighbors$(t_i, e_i, +)$
  end
until *no changes in any substitute*$[x]$
$C_{\text{filtered}} \leftarrow$ substitute$[x_{\text{target}}]$
return C_{filtered}

1. **LLM Cypher Generation:** The LLM generates a Cypher query q_{cypher} from q using node and edge types.
2. **Regex Parsing:** Parse q_{cypher} into target y, triplets T, and symbol constraints S_{raw}.
3. **Symbolic Grounding:**
   - **SYMBOL_CANDIDATES:** Perform VSS for symbol constants.
   - **GROUND_TRIPLETS:** Iteratively prune candidate sets via neighbor intersections.
4. **Adaptive Candidate Expansion:** While $|C_{\text{cypher}}| < k$, exponentially increase $\hat{k}$ and repeat Step 3.
5. **Hybrid VSS Retrieval:** Perform two VSS calls:
   - Y_{cypher}: Cypher-constrained candidates
   - Y_{vss}: Fallback unstructured candidates
6. **LLM Reranking:** Rerank concatenated candidates $Y = Y_{\text{cypher}} + Y_{\text{vss}}$.

GROUND_TRIPLETS Algorithm:
Runtime Analysis:

- **Step 1:** $O(1)$ - single LLM call.

Algorithm 2: GROUND_TRIPLETS: Iterative Candidate Pruning

Input: Triplets $T = \{(h_i, e_i, t_i)\}$, symbol candidate sets S
Output: Pruned symbol candidate sets S
repeat
  foreach $(h_i, e_i, t_i) \in T$ **do**
    $S(h_i) \leftarrow$
     $S(h_i) \cap \text{neighbors}(h_i, e_i, -)$
    $S(t_i) \leftarrow S(t_i) \cap \text{neighbors}(t_i, e_i, +)$
  end
until *no changes in any* $S(x)$
return S

- **Steps 2–4:** $O(n_1 \cdot n_2 \cdot |T| \cdot \deg)$ - n_1 outer $\hat{k}$ expansions, n_2 inner convergence iterations, $|T|$ triplets, $\deg$ average degree.
- **Step 5:** $O(|C_{\text{cypher}}| + |C(y'.\text{type})|)$ - two VSS calls.
- **Step 6:** $O(1)$ - single LLM call for reranking.

Total Complexity:

$$O(n_1 \cdot n_2 \cdot |T| \cdot \deg + |C_{\text{cypher}}|)$$

F.3 AvaTaR

AvaTaR is an agent optimization framework that improves tool-calling LLMs through contrastive reasoning. It is typically instantiated as a ReAct-style agent operating over retrieval tools.

Pipeline:

1. **Agent Initialization:** Initialize the agent with query q and a toolset (e.g., retrieval APIs over an SKB).
2. **Agent Loop:** Repeat for m steps:
   - The LLM generates a *thought* and an *action* (tool call).
   - Execute the action and append the resulting observation to the trajectory.
3. **Termination:** Continue until maximum steps, success condition, or a final answer is generated.
4. **AvaTaR Optimization (Training-time):** Perform contrastive fine-tuning using successful vs. failed trajectories to improve future performance.

Runtime Analysis:

- Per trajectory: $O(m)$ LLM calls and $O(m_r)$ retrievals (m = total steps, $m_r \leq m$ = number of retrieval actions).
- No explicit set operations; reasoning is implicit in the LLM.
- No dedicated reranker; retrieval results are fed directly to the agent prompt.
- Inference can be amortized over multiple queries via the trained policy.

Total Complexity (per query):

$$O(m) \text{ LLM calls} + O(m_r) \text{ retrievals}$$

where m is the trajectory length until stopping condition.

F.4 KAR

KAR performs knowledge-aware query expansion through entity extraction, neighbor retrieval, document relation filtering, triple construction, and final retrieval.

Pipeline:

1. **Entity Extraction:** The LLM extracts entities $E = \{e_1, \ldots, e_e\}$ from query q.
2. **Neighbor and Document Retrieval:** For each entity e_i, retrieve neighbors $N(e_i)$ and associated documents D_i (total $e + 1$ retrievals).
3. **Document Relation Filtering:** Filter retrieved documents based on cosine similarity over the $e + 1$ neighbor sets.
4. **Triple Construction:** Construct triples (e_i, r_j, d_k) from the filtered neighbor sets.
5. **Query Expansion:** Feed all triples to the LLM to generate an expanded query q_{exp}.
6. **Final Retrieval:** Execute a single retrieval over the expanded query q_{exp}.

Runtime Analysis:

- Step 1: $O(1)$ - single LLM call for entity extraction.
- Step 2: $O(e)$ - e entity neighbor retrievals plus 1 document retrieval.

- Step 3: $O((e+1) \cdot \deg_D)$ - cosine similarity over neighbor documents, $\deg_D$ = average doc neighbors.
- Step 4: $O(|T|)$ - triple construction, where $|T|$ is total triples.
- Step 5: $O(1)$ - single LLM call for query expansion.
- Step 6: $O(1)$ - single final retrieval.

Total Complexity:

$$O(e \cdot \deg_D + |T|)$$

with exactly two LLM calls.

F.5 ReAct

ReAct interleaves LLM reasoning with tool calls (typically retrieval) in a think-act-observe loop until task completion or a step limit.

Pipeline:

1. **Initial LLM Reasoning:** The LLM generates an initial thought and first action (retrieval tool call).
2. **Execute Action:** Execute the retrieval action and append results as an observation to the trajectory.
3. **Subsequent Reasoning:** LLM reasons over the observation and generates the next thought and action.
4. **Iteration:** Repeat Steps 2–3 for m iterations until a final answer is obtained or maximum steps are reached.

Runtime Analysis:

- Per trajectory: $O(m)$ LLM calls and $O(m_r)$ retrievals.
- m: total steps until stopping condition (max steps, success, or final answer).
- $m_r \leq m$: number of retrieval actions (subset of total actions).
- No explicit set operations; reasoning is implicit in the LLM.
- No dedicated reranker; retrievals are fed directly to the agent prompt.

Total Complexity (per query):

$$O(m) \text{ LLM calls} + O(m_r) \text{ retrievals}$$

where m is the trajectory length until stopping.

F.6 Reflexion

Reflexion extends ReAct agents with self-reflection: after each failed episode, an LLM analyzes the trajectory to generate verbal feedback for the next episode.

Pipeline:

- **Step 1:** Run ReAct episode: m iterations of LLM call $\rightarrow$ retrieval $\rightarrow$ observation.
- **Step 2:** Evaluate episode outcome (success/-failure).
- **Step 3:** If failed, LLM generates reflection r_i analyzing trajectory and errors.
- **Step 4:** Append reflection to prompt, repeat Steps 1–3 for E episodes total.

Runtime analysis:

- Per episode: $O(m)$ LLM calls + $O(m_r)$ retrievals (m: steps, m_r: retrievals).
- Per reflection: $O(1)$ LLM call (self-critique).
- E: total episodes until success or cap.
- R: reflections ($R \leq E - 1$, one per failed episode).
- Total: $O(E \cdot m + R)$ LLM calls + $O(E \cdot m_r)$ retrievals.

Total: $O(E \cdot m + R)$ LLM calls + $O(E \cdot m_r)$ retrievals, where E is episodes, m is steps per episode, R is reflections.

F.7 SAGE using SPARK

SAGE using SPARK queries through a single-pass agentic retrieval pipeline with hybrid dense+sparse expansion.

Pipeline:

1. **Query Generation:** Agent LLM generates a Cypher/HNSW query from the natural language input q.
2. **Initial Retrieval:** Retrieve the top-k nodes N_k using the generated query.
3. **Graph Expansion:** Retrieve 1-hop neighbors $N(N_k)$ of the top-k nodes.
4. **Hybrid Retrieval:** Apply dense+sparse retrieval over $N(N_k)$ to produce the final $k + k'$ candidate set.

Runtime Analysis:

- Step 1: $O(1)$ - single agent LLM call.
- Step 2: $O(k)$ - initial retrieval (HNSW/-Cypher).
- Step 3: $O(k \cdot \deg)$ - neighbor expansion ($\deg$ = average node degree).
- Step 4: $O(k \cdot \deg)$ - hybrid dense+sparse retrieval over neighbors.

Total Complexity:

$$O(k \cdot \deg)$$

with exactly one LLM call and no iterative loops.

Latency is a potential concern when combining graph and vector retrieval; however, the system is designed to mitigate this through parallel execution of both components. Graph traversal is performed using Neo4j, while vector retrieval relies on an HNSW index, with both operations typically completing within milliseconds in practice. The dominant latency arises not from database access but from Cypher query generation using GPT-4o-mini, which averages approximately 0.5 seconds. This step is deliberate, as it converts the user query into structured graph constraints, enabling targeted graph expansion rather than unguided neighborhood exploration.

G Effect of Dense+Sparse Retriever + Graph Expansion on STaRK Dataset

G.1 AMAZON

Figures 13 and 14 show recall@k for Synthetic and Human queries on the AMAZON subset. Graph expansion consistently improves recall across all k values. This aligns with our observation that nodes in AMAZON contain rich product descriptions and feature-rich metadata, providing ample semantic content for propagation. The one-hop expansion strategy reliably increases coverage of relevant nodes without introducing noise, yielding a consistent gain over the Dense+Sparse baseline.

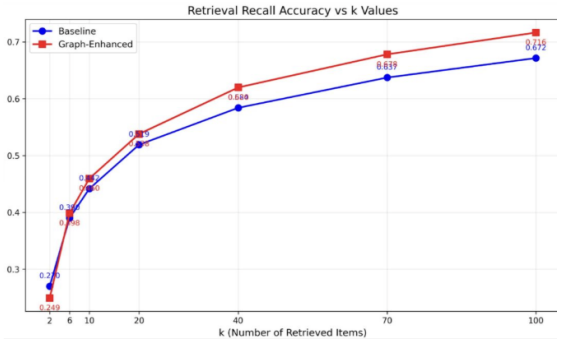

Figure 13: Recall@k for Synthetic queries on AMAZON. Graph expansion improves recall across all k.

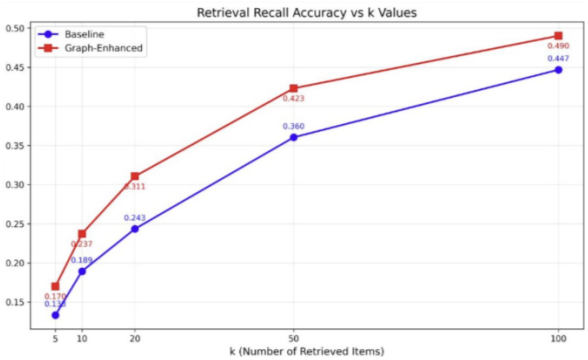

Figure 14: Recall@k for Human queries on AMAZON. Graph expansion consistently increases recall.

G.2 MAG

Figures 15 and 16 show recall@k for MAG. Here, graph expansion helps primarily at higher k values ($k \gtrsim 10$), while recall may drop at lower k. MAG nodes contain shorter titles and abstracts, so expanding the graph at small k can introduce weakly relevant neighbors, hurting early-rank retrieval. At larger k, however, graph augmentation still provides additional relevant nodes, boosting recall and demonstrating its utility when more candidates are available.

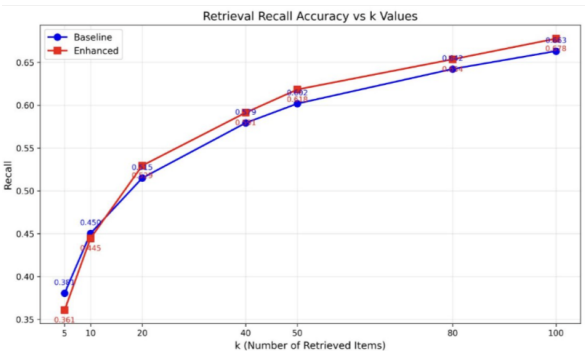

Figure 15: Recall@k for Synthetic queries on MAG. Graph expansion improves recall at higher k, but hurts at low k.

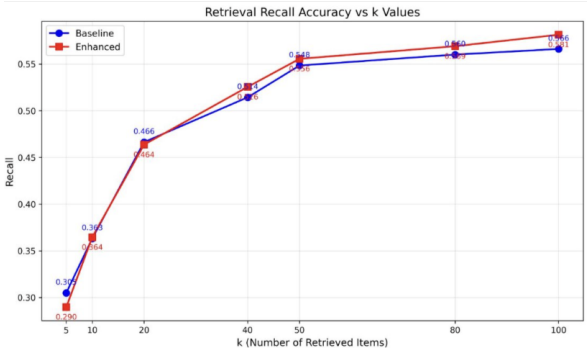

Figure 16: Recall@k for Human queries on MAG. Graph expansion shows gains only at larger k.

G.3 PRIME

Figures 17 and 18 show recall@k for PRIME. Similar to MAG, graph expansion benefits are observed primarily at higher k values ($k \gtrsim 20$), while lower k sees slight drops in recall. PRIME nodes contain concise drug, pathway, and disease descriptions, limiting semantic propagation. These trends reinforce our finding that graph-based expansion is most effective when nodes have rich textual content.

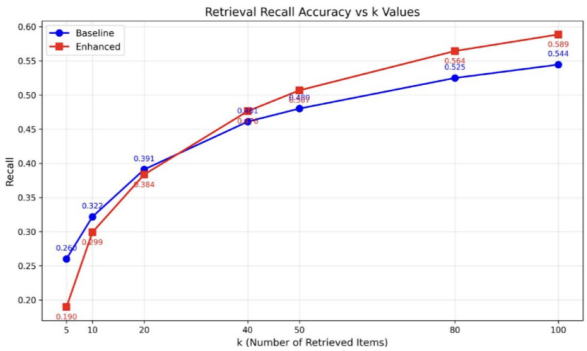

Figure 17: Recall@k for Synthetic queries on PRIME. Graph expansion is effective only at higher k.

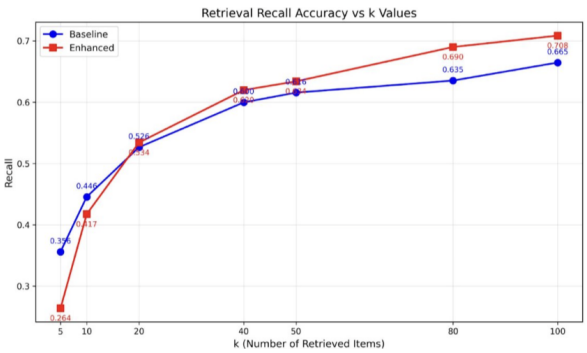

Figure 18: Recall@k for Human queries on PRIME. Graph expansion helps mainly at larger k.

H Implementation Details.

Our system follows a two-stage design with offline preprocessing and online query processing. Offline, we perform semantic chunking and graph construction using the all-MiniLM-L6-v2 embedding model (384-d), (Wang et al., 2020) and build ANN indices with HNSW (M=64); preprocessing is executed once on a single NVIDIA A100 GPU. For explicit schema graphs, we store and query the structured data in Neo4j, and extract lightweight metadata such as titles and entities from nodes/fields to support linking and retrieval. Online, we answer each query by combining sparse and dense signals via linear fusion ($\alpha = 0.6$ for BM25, $\beta = 0.4$ for embeddings), and we use GPT-4o-mini-2024-07-18 (OpenAI et al., 2024) where LLM-based extraction/planning is required.

I Scalability of Graph Construction

Naively computing full pairwise similarities between all nodes scales quadratically, $O(N^2)$, which becomes infeasible for very large corpora. To address this, similarity computation can be restricted to an approximate nearest neighbor (ANN) candidate pool rather than the entire corpus.

For each node, the nearest K candidates (e.g., 10K–30K) can be retrieved using HNSW, and exact similarities computed only within this neighborhood. The 95th percentile pruning is then applied locally over these candidates. This reduces graph construction from $O(N^2)$ to $O(NK)$ for similarity evaluation, where $K \ll N$, making large-scale offline graph construction practical.

To evaluate whether such ANN-restricted neighborhoods preserve relevant connections, we analyze recall at different candidate sizes on the STARK-AMAZON corpus, our largest dataset. Table 22 reports Hit@K and Recall@K as the candidate pool grows.

K	Hit@K	Recall@K
100	0.716	0.321
1,000	0.840	0.605
5,000	0.926	0.806
10,000	0.938	0.863
20,000	0.988	0.935
30,000	1.00	1.00

Table 22: Recall of relevant neighbors under ANN-restricted candidate pools on the STARK-AMAZON corpus.

The high recall and hit rates indicate that most relevant neighbors are preserved within these candidate sets. These results suggest that large-scale graph construction can rely on ANN-local neighborhoods without materially degrading graph quality while avoiding quadratic computational cost.

J Additional Metrics for STaRK

Table 23 presents Hits@1, Hits@5, and MRR scores for all methods across AMAZON, MAG, and PRIME datasets.

Method	Amazon						MAG						Prime					
	Human			Synthetic			Human.			Synthetic			Human.			Synthetic		
	H1	H5	M	H1	H5	M	H1	H5	M	H1	H5	M	H1	H5	M	H1	H5	M
Dense																		
ada-002	39.50	64.19	52.65	29.08	49.61	38.62	28.57	41.67	35.81	29.08	49.61	38.62	17.35	34.69	26.35	12.63	31.49	21.41
m-ada	46.91	72.84	58.74	40.07	64.98	51.55	23.81	41.67	31.43	25.92	50.43	36.94	24.49	39.80	32.98	15.10	33.56	23.49
DPR	16.05	39.51	27.21	15.29	47.93	30.20	4.72	9.52	7.90	10.51	35.23	21.34	2.04	9.18	7.05	4.46	21.85	12.38
Sparse																		
BM25	27.16	51.85	18.79	25.85	45.25	34.91	32.14	41.67	37.42	25.85	45.25	34.91	22.45	41.84	30.37	12.75	27.92	19.84
Structural																		
QAGNN	22.22	49.38	31.33	12.88	39.01	29.12	20.24	26.19	25.53	12.88	39.01	29.12	6.12	13.27	26.35	8.85	21.35	14.73
Hybrid																		
4Step	–	–	–	47.60	67.60	56.50	–	–	–	53.80	69.20	61.40	–	–	–	39.30	53.20	45.80
FocusR	–	–	–	64.00	76.20	69.30	–	–	–	74.10	84.20	78.80	–	–	–	46.40	63.90	53.70
D+B	32.14	44.05	38.46	29.72	49.64	38.71	32.14	44.05	38.46	29.72	49.64	38.71	24.77	43.12	33.25	13.71	31.13	21.39
Query Expansion																		
HyDe	45.68	72.84	57.56	28.98	50.10	39.58	33.33	44.05	38.95	28.98	50.10	39.58	24.77	42.20	33.65	16.85	37.59	26.56
RAR	55.56	71.60	62.15	39.02	52.87	39.58	38.10	45.24	42.04	39.02	52.87	39.58	31.19	43.12	37.72	22.23	40.84	30.93
AGR	55.56	71.60	63.54	39.29	53.66	46.20	33.33	44.05	38.95	39.29	53.66	46.20	32.11	49.54	39.27	25.85	44.41	35.04
KAR	61.73	72.84	66.32	50.47	65.37	57.51	51.20	58.30	54.20	50.47	65.37	57.51	44.95	60.55	44.51	30.35	49.30	39.22
Agentic																		
AvaTaR	58.32	76.54	65.91	49.97	69.16	52.01	33.33	42.86	38.62	46.08	59.32	52.01	33.03	51.37	41.00	20.10	39.89	29.18
ReACT	45.65	71.73	58.81	42.14	64.56	52.30	27.27	40.00	33.94	31.07	49.49	39.25	21.73	33.33	28.20	15.28	31.95	22.76
Reflex	49.38	64.19	52.91	42.79	65.05	52.91	28.57	39.29	36.53	40.71	54.44	47.06	16.52	33.03	23.99	14.28	34.99	24.82
Finetuned																		
mFAR	–	–	–	41.20	70.00	54.20	–	–	–	49.00	69.60	58.20	–	–	–	40.90	62.80	51.20
MoR	–	–	–	52.20	74.60	62.20	–	–	–	58.20	78.30	67.10	–	–	–	36.40	60.00	46.90
Ours																		
SPARK	22.22	58.03	39.93	26.72	54.27	36.24	44.40	54.80	49.40	41.00	56.41	49.30	41.66	61.14	52.53	29.12	48.31	41.94
A+G+E	25.18	55.13	37.75	25.18	53.45	36.18	45.10	55.60	48.90	41.70	57.20	48.80	42.20	61.60	51.53	30.14	49.56	42.79
A+G-E	28.41	63.84	42.48	28.41	55.13	42.48	43.20	58.70	49.40	43.20	60.00	49.30	40.23	65.40	52.53	30.86	51.48	43.20

Table 23: Hits@1 (H@1), Hits@5 (H@5), and Mean Reciprocal Rank (MRR) scores on STaRK datasets. A = SPARK, G=SAGE, E=Edge

K Prompt for LLM Metadata Extraction

K.1 Entity Extraction for Table Chunks

Listing 1: Table metadata extraction prompt

```
You are a table metadata extraction
    engine. Given table data with
    headers and rows, parse it into a
    single valid JSON object following
    exactly this schema:

 table_title ,<verbatim table title or
    inferred title if unavailable>
 table_description ,<one-sentence
    explanation of the table content>
 table_summary ,<concise interpretation
    or insight derived from the data>
 col_desc ,
  <ColumnName1>,<contextual
      description of the column's
      representation and purpose>
 col_format ,
  <ColumnName1>,<data format
      specification,string,number,date
      (YYYY-MM-DD),etc.>
 entities ,
  <EntityName1>,
    type,<person|place|organization|
        event|concept|thing>
    category,<named|non-named>
    description,<brief description of
        the entity>

 Instructions:
1.  table_title : Use the source name if
     available; otherwise, infer a
    descriptive title from the data.
2.  table_description : Provide one
    sentence explaining the information
     contained in this table.
3.  table_summary : Provide a brief
    insight or interpretation of the
    overall data patterns.
4.  col_desc : For each column header,
    provide a contextual description of
     what the column represents, its
    purpose, and significance.
5.  col_format : For each column header,
     specify the data format (e.g.,
    string - names, number - scores,
    date (Month Day)).
6.  entities : Extract all unique
     entities mentioned in the table
    with detailed information:
   - type: person, place, organization
       , event, concept, thing, etc.
   - category: named (proper nouns) or
       non-named (common nouns/
       concepts)
   - description: brief explanation of
       what this entity represents

Output: Pure JSON only. No markdown or
     comments.

One-Shot Example :

Table Source Name: "1911 Notre Dame
    Fighting Irish football team -
    Schedule"
Table Data:
[
  {{
    "Date": "October 7",
    "Opponent": "Ohio Northern",
    "Site": "Cartier Field South Bend
        , IN",
    "Result": "W 32-6"
  }},
  {{
    "Date": "October 14",
    "Opponent": "St. Viator",
    "Site": "Cartier Field South Bend
        , IN",
    "Result": "W 43-0"
  }},
  {{
    "Date": "October 21",
    "Opponent": "Butler",
    "Site": "Cartier Field South Bend
        , IN",
    "Result": "W 27-0"
  }}
]

Expected Output:
 table_title ,1911 Notre Dame Fighting
    Irish football team - Schedule
 table_description ,This table presents
    the football schedule and results
    for the 1911 Notre Dame Fighting
    Irish football team.
 table_summary ,Notre Dame had a strong
    season with multiple wins at home
    field,scoring consistently high
    points against various opponents.
 col_desc ,
  Date,The scheduled date when each
      football game was played during
      the 1911 season
  Opponent,The opposing football team
      that Notre Dame played against in
       each game
  Site,The venue and location where
      each game took place,including
      home and away games
  Result,The final outcome and score
      of each game showing Notre Dame's
       performance
 col_format ,
  Date,string - date (Month Day format
      )
  Opponent,string - team names
  Site,string - locations with venue
       details
  Result,string - game results (W/T/L
      followed by score)
 entities ,
  Notre Dame Fighting Irish,
    type,organization
    category,named
    description,College football team
        from the University of Notre
        Dame
  Ohio Northern,
    type,organization
    category,named
```

```
    description,Opposing college
        football team
  St. Viator,
    type,organization
    category,named
    description,Opposing college
        football team
  Butler,
    type,organization
    category,named
    description,Opposing college
        football team
  Cartier Field,
    type,place
    category,named
    description,Notre Dame's home
        football stadium
  South Bend,
    type,place
    category,named
    description,City in Indiana where
        Notre Dame is located
  IN,
    type,place
    category,named
    description,Indiana state
        abbreviation

Now, analyze this table data:

Table Source Name: "{ source_name }"
Table Data:
{ table_content }
```

K.2 Entity Extraction for Document Chunks

Listing 2: Document metadata extraction prompt

```
You are a metadata extraction engine.
    Given a block of text, parse it
    into a single valid JSON object
    following exactly this schema:

 entities ,
   <EntityName1>,
     details [1],<detailed attribute or
        description sentence>
 events ,
   <EventName1>,
     date,<YYYY-MM-DD or null>
     details ,<one-sentence context or
        significance>
 timeline [1],<ISO date - description>
 topic ,<one-sentence description of the
     overall content>

 Instructions:
1.  entities : One key per unique NAMED
    entity (not a noun phrase).
    Nicknames and abbreviations count
    as separate  entities , but the
    original must be mentioned in
    details  (e.g., ORIGINAL: ...).
   -  details : Distinguishing
       descriptive sentences or
       attributes where the entity is
       the subject.
2.  events : One key per named event.
   - date: ISO date or null.
   -  details : A single-sentence
       summary of its significance.
3.  timeline : Chronological array of "<
    date> - <brief description>" for
    all dated mentions.
4.  topic : One sentence summarizing the
     overall theme.

Output: Pure JSON only. No markdown or
     comments.
Note: Each object must have unique
    keys.

OneâĂŘShot  Example

Input Text

Bixente Jean Michel Lizarazu (Basque
    pronunciation: [biËĹÊČente
    liËĹsÌłaraËŇsÌłu], born 9 December
    1969) is a French former
    professional footballer who played
    as a left-back.

He rose through the ranks at Bordeaux
    and finished second in the French
    First Division in 1989-1990. The
    team was relegated but won
    promotion from Second Division in
    1991-92. His Bordeaux team finished
     runners-up in the 1995-96 UEFA Cup
    .

In 1997, he joined Bayern Munich and
    won six Bundesliga championships
    and the 2000-01 UEFA Champions
    League, scoring in the final
    shootout.

Expected Output
 entities ,
   Bixente Jean Michel Lizarazu,
   Bordeaux FC,

   Bayern Munich,

   Bundesliga,

   France,

   French First Division,

   French Second Division,

 events ,
   1989-1990 French First Division
       season,
     date,1989-1990
   1995-1996 UEFA Cup Final,
     date,1996-05-08

   1997 Transfer to Bayern Munich,
     date,1997-07-01

   2001 UEFA Champions League Final,
     date,2001-05-23


 topic ,Club career progression and
    major achievements of Bixente
```

```
    Lizarazu across France and Germany,
    including French and German
    domestic leagues

CRITICAL REQUIREMENTS:
- Include EVERY entity or event
    mentioned directly or indirectly in
     the text.
- Ensure all delimiters and quotes are
     correctly placed.
- Verify that your output is valid
    JSON.
- Use only the specified keys; do not
    add additional keys.
- Output a JSON object only (no
    variable assignments or equal signs
    ).

Given any new text chunk, output
    exactly this JSON structure with
    all fields populated from the text.



This is the text you must parse and
    provide metadata for:
Document Title: "{ doc_title }"
Document Content: { doc_content }
```

L Prompts for Agentic baseline

L.1 AMAZON

Listing 3: Neo4j prompt for schema-aware retrieval on Amazon graph

```
### Neo4j Graph Database (Write Cypher
     Queries Directly)
For ALL graph traversal, write Cypher
    queries directly. Do NOT use
    function calls for graph traversal.

All graph data is in Neo4j. You must:
1) Understand the question: which
     entities and relationships are
    involved?
2) Plan the traversal: which node
    labels, relationships, and
    constraints are needed?
3) Write Cypher: output ONE Cypher
    query that returns the answer
    candidates.

Neo4j graph schema:
(Product)-[:HAS_REVIEW]->(Review)
                Products have customer
    reviews
(Product)-[:ALSO_BUY]-(Product)
                 Co-purchase
    recommendations
(Product)-[:ALSO_VIEW]-(Product)
                Co-view recommendations

Product properties (selected):
- p.nodeId (int, unique)
- p.asin (str, unique)
    Amazon Standard Identification
    Number
- p.title (str)
- p.brand (str)
- p.price (str)
    String format: "$11.80"
- p.color (str/list)
- p.globalCategory (str)
    High-level category
- p.category (list[str])
    Specific categories
- p.feature (list[str])
    Product features/specifications
- p.description (list[str])
    Product descriptions
- p. details (str)
    JSON-encoded specifications
- p.rank (str)
    Sales rank

Review properties (selected):
- r.nodeId (int, unique)
- r.asin (str)
    Product ASIN
- r.reviewerID (str)
- r.overall (float)
    Star rating (1.0 to 5.0)
- r.reviewText (str)
- r.summary (str)
    Review title
- r.verified (bool)
    Verified purchase
- r.reviewTime (str)
    Review date
- r.vote (str)
    Helpful vote count
- r.style (str)
    Product variant info (JSON)

Cypher examples:

 Example 1: Products by brand in a
    category
MATCH (p:Product)
WHERE toLower(p.brand) CONTAINS 'nike'
RETURN p.nodeId

 Example 2: Products with specific
    features
MATCH (p:Product)
WHERE toLower(p.title) CONTAINS '
    backpack'
  AND (toLower(p.feature) CONTAINS '
      waterproof'
       OR toLower(p.description)
           CONTAINS 'waterproof')
RETURN p.nodeId

 Example 3: Products with specific
    rated reviews, sorted by average
    rating
MATCH (p:Product)-[:HAS_REVIEW]->(r:
    Review)
WHERE toLower(p.title) CONTAINS '
    baseball cap'
  AND r.overall >= 4.0
RETURN p.nodeId, avg(r.overall) AS
    avgRating, count(r) AS reviewCount
ORDER BY avgRating DESC, reviewCount
    DESC

 Example 4: Products under a price
    threshold
```

```
MATCH (p:Product)
WHERE toLower(p.title) CONTAINS 'knife
    '
  AND toFloat(substring(p.price, 1)) <
       20.0
RETURN p.nodeId

 Example 5: Recommendations based on co
    -purchase
MATCH (bought:Product)-[:ALSO_BUY]-(p:
    Product)
WHERE bought.asin = '0000032042'
  AND toLower(p.title) CONTAINS '
      accessories'
RETURN DISTINCT p.nodeId
```

L.2 MAG

Listing 4: Neo4j prompt for schema-aware retrieval

```
### Neo4j Graph Database (Write Cypher
     Queries Directly)
For ALL graph traversal, write Cypher
    queries directly. Do NOT use
    function calls for graph traversal.

All graph data is in Neo4j. You must:
1) Understand the question: which
     entities and relationships are
    involved?
2) Plan the traversal: which node
    labels, relationships, and
    constraints are needed?
3) Write Cypher: output ONE Cypher
    query that returns the answer
    candidates.

Neo4j graph schema:
(Author)-[:AUTHORED]->(Paper)
                    Authors write
    papers
(Paper)-[:CITES]->(Paper)
                        Citation
    network
(Paper)-[:HAS_FIELD]->(Field)
                    Fields of study
(Author)-[:AFFILIATED_WITH]->(
    Institution)    Author affiliations

Paper properties (selected):
- p.paperId (int, unique)
- p.title (str)
- p.abstract (str)
- p.year (int)
- p.date (str)  Use p.date for exact
    date constraints (e.g.,
    '2016-02-11')
- p.journalDisplayName (str)
- p.docType (str)
- p.paperCitationCount (int)  Use for
    "most cited" constraints

Author properties (selected):
- a.authorId (int, unique)
- a.name (str)
- a.displayName (str)

Field properties (selected):
- f.fieldId (int, unique)
- f.name (str)

Institution properties (selected):
- i.institutionId (int, unique)
- i.name (str)

Cypher examples:

 Example 1: Papers by an author in a
    given year
MATCH (:Author {authorId: 324400})-[:
    AUTHORED]->(p:Paper)
WHERE p.year = 2017
RETURN p.paperId

 Example 2: Papers by exact date (use p
    .date, not only p.year)
MATCH (p:Paper)
WHERE p.date = '2016-02-11'
RETURN p.paperId

 Example 3: Most cited paper matching a
     textual constraint
MATCH (p:Paper)
WHERE toLower(p.title) CONTAINS
    toLower('graph retrieval')
   OR toLower(p.abstract) CONTAINS
       toLower('graph retrieval')
RETURN p.paperId, p.paperCitationCount
ORDER BY p.paperCitationCount DESC
LIMIT 1
```

L.3 PRIME

Listing 5: Neo4j prompt for schema-aware retrieval on PRIME graph

```
### Neo4j Graph Database (Write Cypher
     Queries Directly)
For ALL graph traversal, write Cypher
    queries directly. Do NOT use
    function calls for graph traversal.

All graph data is in Neo4j. You must:
1) Understand the question: which
     entities and relationships are
    involved?
2) Plan the traversal: which node
    labels, relationships, and
    constraints are needed?
3) Write Cypher: output ONE Cypher
    query that returns the answer
    candidates.

Neo4j graph schema:
(Gene)-[:INTERACTS_WITH_PROTEIN]-(Gene
    )               Protein-protein
    inter actions
(Drug)-[:TARGETS]->(Gene)
                                Drug
    targets gene/protein
(Drug)-[:INDICATED_FOR]->(Disease)
                      Drug treats
    disease
(Drug)-[:CONTRAINDICATED_FOR]->(
    Disease)              Drug should
    NOT be used
(Drug)-[:SYNERGISTIC_WITH]-(Drug)
                        Drug synergy
```

```
(Drug)-[:HAS_SIDE_EFFECT]->(Phenotype)
                    Drug side effects
(Gene)-[:ASSOCIATED_WITH]-(Disease)
                      Gene-disease
    associations
(Gene)-[:EXPRESSED_IN]->(Anatomy)
                        Gene expression
    in tissue
(Gene)-[:INTERACTS_WITH]->(Pathway)
                      Gene in pathway
(Gene)-[:INTERACTS_WITH]->(
    MolecularFunction)        Gene
    function
(Gene)-[:INTERACTS_WITH]->(
    BiologicalProcess)        Gene in
    process
(Disease)-[:PHENOTYPE_PRESENT]->(
    Phenotype)          Disease
    symptoms
(Disease)-[:LINKED_TO]->(Exposure)
                      Disease
    environmental links

Node properties (selected):
Gene:
- g.nodeId (int, unique)
- g.name (str)
- g.sourceId (str)
- g.details Json (str)
    Contains: summary, aliases,
    location

Disease:
- d.nodeId (int, unique)
- d.name (str)
- d.sourceId (str)
- d.details Json (str)
    Contains: definition, symptoms

Drug:
- drug.nodeId (int, unique)
- drug.name (str)
- drug.sourceId (str)
- drug.details Json (str)
    Contains: mechanism, indication

Phenotype:
- p.nodeId (int, unique)
- p.name (str)

Pathway:
- pw.nodeId (int, unique)
- pw.name (str)
- pw.stId (str)
    Reactome stable ID
- pw.details Json (str)

Anatomy:
- a.nodeId (int, unique)
- a.name (str)

MolecularFunction, BiologicalProcess,
    CellularComponent:
- nodeId, name (standard properties)

Cypher examples:

 Example 1: Disease with multiple
    phenotypes
MATCH (d:Disease)-[:PHENOTYPE_PRESENT
    ]->(p:Phenotype)
WHERE p.name IN ['pharyngitis', '
    chemosis']
RETURN DISTINCT d.nodeId, d.name

 Example 2: Drug for a disease
MATCH (drug:Drug)-[:INDICATED_FOR]->(d
    :Disease)
WHERE d.name = 'sclerosing cholangitis
    '
RETURN drug.nodeId, drug.name

 Example 3: Gene with protein-protein
    interactions
MATCH (g:Gene)-[:
    INTERACTS_WITH_PROTEIN]-(g2:Gene)
WHERE g2.name IN ['hbq1', 'sirt5']
RETURN DISTINCT g.nodeId, g.name

 Example 4: Gene in pathway and
    biological process
MATCH (g:Gene)-[:INTERACTS_WITH]->(bp:
    BiologicalProcess)
MATCH (g)-[:INTERACTS_WITH]->(pw:
    Pathway)
WHERE bp.name = 'cellular response to
    manganese ion'
  AND toLower(pw.name) CONTAINS 'atp'
RETURN g.nodeId, g.name

 Example 5: Drug with target and
    contraindication
MATCH (drug:Drug)-[:TARGETS]->(g:Gene)
MATCH (drug)-[:CONTRAINDICATED_FOR]->(
    d:Disease)
WHERE g.name = 'ccr5'
  AND d.name = 'gout'
RETURN drug.nodeId, drug.name
```