SAGE: Structure Aware Graph Expansion for Retrieval of Heterogeneous Data

Anonymous ACL submission

Abstract

Retrieval-augmented question answering over heterogeneous corpora requires *connected* evidence across text, tables, and graph nodes. While entity-level knowledge graphs support structured access, they are costly to construct and maintain, and inefficient to traverse at query time. In contrast, standard retriever-reader pipelines use flat similarity search over independently chunked text, missing multi-hop evidence chains across modalities. We propose SAGE (**S**tructure **A**ware **G**raph **E**xpansion) framework that (i) constructs a chunk-level graph *offline* using metadata-driven similarities with percentile-based pruning, and (ii) performs *online* retrieval by running an initial baseline retriever to obtain k seed chunks, expanding first-hop neighbors, and then filtering the neighbors using dense+sparse retrieval, selecting k' additional chunks. We instantiate the initial retriever using hybrid dense+sparse retrieval for implicit cross-modal corpora and SPARK (**S**tructure Aware **P**lanning **A**gent for **R**etrieval over **K**nowledge Graphs) an agentic retriever for explicit schema graphs. On OTT-QA and STaRK, SAGE improves retrieval recall by 5.7 and 8.5 points over baselines.

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.

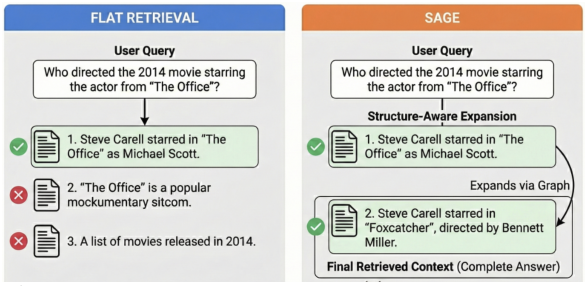

Figure 1: Illustration of SAGE 's seed→expand retrieval: starting from an initial relevant chunk, graph expansion retrieves a connected neighbor containing the missing movie/director evidence that flat retrieval fails to surface.

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 domains 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 in-

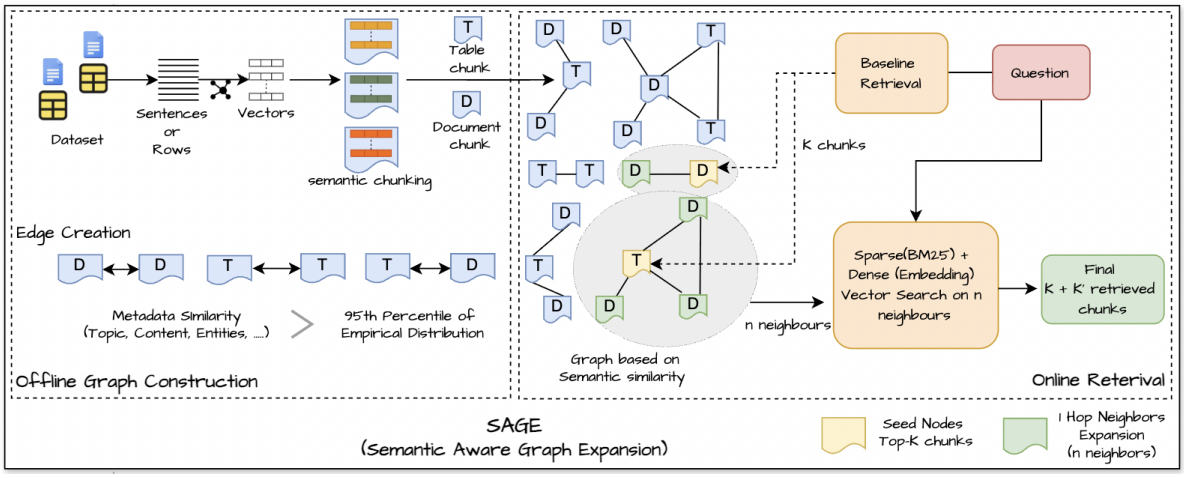

Figure 2: Overview of SAGE. **Offline:** we semantically chunk documents and tables into nodes and create edges using metadata-driven similarity (e.g., title, topic, content, entities) with percentile-based pruning. **Online:** a baseline retriever returns k seed nodes, we expand to n first-hop neighbors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

troduce spurious neighbors, while overly sparse graphs hinder multi-hop reasoning. Node granularity also 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 propose a chunk-level graph to improve recall through structure-aware expansion over semantically coherent units. Instead of retrieving many independent chunks, a small set of seed nodes is expanded via graph traversal, surfacing evidence weakly similar to the query but strongly connected via document structure, metadata, or shared context. Nodes represent text or table segments, preserving coherence without explicit entity extraction. Retriever-agnostic and built offline, the graph adds minimal overhead, while compact neighborhood traversal supports efficient multi-hop reasoning and reliable evidence retrieval across heterogeneous corpora. Our contributions are as follows:

- We introduce SAGE, a **structure-aware retrieval framework** that uses a chunk-level graph to support neighbor expansion beyond flat retrieval, while remaining compatible with existing retrievers.
- SAGE demonstrates that **similarity-induced graph expansion** boosts retrieval recall over strong flat hybrid baselines under fixed budgets (+5.7 OTT-QA, +8.5 STaRK), especially for implicit multi-hop/cross-modal queries.
- **Schema-constrained expansion** via SPARK, our retrieval agent, enforces structural validity in explicit schema graphs during neighbor traversal, outperforming naive expansion in dense settings.

2 SAGE Approach

Given a corpus represented as a graph $G = (V, E)$ with node content $\mathrm{text}(v)$ and optional metadata $\mathrm{meta}(v)$, 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 a sequence of sentences and embed overlapping context windows $w_i = [s_{i-1}, s_i, s_{i+1}]$ (with boundary truncation) 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 as $s_i = \cos(\mathbf{h}_i, \mathbf{h}_{i+1})$.

We introduce a boundary at indices where local coherence drops, using a percentile-based rule over $\{s_i\}$: letting $\theta = \mathrm{Percentile}_{20}(\{s_i\})$ (equivalently, selecting the top-80% "drop" positions), we split

at all i such that $s_i \leq \theta$. After chunking, we use an LLM to generate per-chunk metadata (topic, title, and entities) for downstream graph construction.

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.

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. We also preserve parent/structural links (e.g., chunks from the same source) and retain explicit schema edges when available. Figure 2 illustrates the resulting graph structure.

Edge metadata for traversal. Edges store lightweight metadata to guide controlled graph traversal. Table–table edges record joinable column names, document–document edges record shared entities and confidence scores, and table–document edges record shared entities and, when available, row or column references. This metadata allows multi-hop reasoning to prioritize neighbors that align with query entities or satisfy structural constraints.

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.

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 entity required by the query.

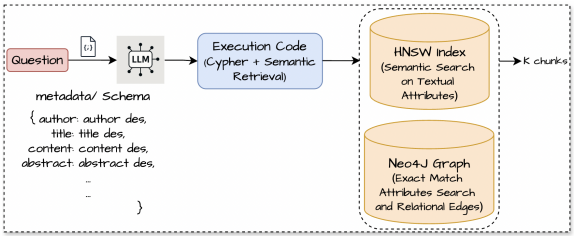

Figure 3: SKB baseline retrieval: the question and schema metadata guide an LLM agent to generate a retrieval plan interleaving HNSW semantic search with Cypher symbolic queries.

To address this, we introduce SPARK, an LLM-based agent for SKB retrieval. Given a question, the agent has access to the entity schemas, relations, and two retrieval tools. It generates a multi-step plan that alternates between: **(1) Semantic Candidate Generation**: retrieving nodes via Hierarchical Navigable Small World (HNSW) (Malkov and Yashunin, 2018) search based on textual similarity and descriptive attributes. This enables flexible matching, allowing the agent to identify semantically relevant nodes even when query keywords do

not appear explicitly. **(2) Structural Expansion**: traversing edges with Cypher queries in Neo4j (Neo4j, Inc., 2026), constrained by entity types and relations to ensure multi-hop structural correctness. This guarantees that retrieved nodes satisfy explicit relational and type constraints, which simple keyword or similarity-based methods cannot enforce.

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 or encounter nodes with sparse textual content. 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 a unified graph expansion strategy to both documents and KGs. Given the k seed nodes from initial retrieval: (1) **Neighbor Expansion:** collect all first-hop neighbors of the k seeds from the pre-built graph G, producing n candidate nodes; (2) **Neighbor Pruning:** re-rank the n candidates using BM25 and dense embedding similarity with respect to the query, and select the top-k' neighbors; (3) **Final Context:** return the union of the original k seeds and the k' neighbors, yielding $k+k'$ total nodes.

Listing 1: Unified $k+k'$ retrieval (pseudo-code)

```
Input: query q, budgets k,k', baseline
     retriever BL, graph G
S = TopK(BL(q), k)
N = OneHopNeighbors(G, S)
R = TopK(BM25DenseRank(q, N), k')
Return Union(S, R)
```

This approach is significantly more efficient than computing node-to-node similarity over the entire graph at runtime. By leveraging pre-computed connections, it captures bridging evidence, reduces noise by focusing on relevant neighbors, preserves multi-hop structural context, and scales to large graphs without the overhead of exhaustive pairwise comparisons.

3 Experiments

Datasets. We evaluate on OTT-QA (Chen et al., 2021) and STaRK (Wu et al., 2024b). OTT-QA is derived from Wikipedia and contains both text and tables, where each table is originally connected to a set of documents. We extract the dev set to build our graph. For this study, we construct the graph from scratch without using the predefined connections. STaRK provides semi-structured KGs spanning three domains: AMAZON (product recommendation with few node types but extensive textual content), MAG (academic citations with papers and authors, supporting one- and two-hop queries), and PRIME (biomedical knowledge with rich relational structure where all nodes are answerable). The AMAZON dataset includes perturbed queries for robustness evaluation. Table 1 summarizes the dataset statistics after graph construction.

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. Node and edge counts are total numbers in the graph. Node and edge types indicate schema diversity.

Further details on node and edge types are in Appendix A .

Baselines for OTT-QA Prior work on OTT-QA mainly targets table-level rather than chunk-level retrieval and typically relies on the dataset's predefined table–document links. Methods that ignore these links, such as ARM (Chen et al., 2025b), use different evaluation settings (full or gold sets), making direct comparison infeasible. BL is a flat hybrid retriever combining BM25 and dense embeddings. Extended BL retrieves the same number of chunks directly with BL, without graph-based expansion.

Baselines for STaRK For STaRK, we compare against a wide range of prior methods. Dense retrievers such as **ada-002**, **multi-ada-002**, and **DPR** (Karpukhin et al., 2020) embed queries and passages into a shared vector space for semantic retrieval. **BM25** serves as a purely sparse, lexical matching baseline. **QA-GNN** (Yasunaga et al., 2021) constructs a joint graph over question entities and KG nodes and performs message passing for relevance scoring. Hybrid approaches like **4Step-Focus** (Boer et al., 2024) and **FocusedR** (Boer et al., 2025) combine vector search with LLM-based triplet extraction and iterative candidate refinement. Query expansion methods such as **HyDE** (Gao et al., 2023), **AGR** (Chen et al., 2024), **RAR** (Shen et al., 2024), and **KAR** (Xia et al., 2024) reformulate or expand queries using KG or LLM-derived information. Agentic systems include **ReACT** (Yao et al., 2022) , which interleaves rea-

soning and retrieval actions, and **Reflexion** (Shinn et al., 2023), which incorporates episodic memory to iteratively improve decisions. Finally, fine-tuned semi-structured retrievers such as **mFAR** (Li et al., 2024) and **MoR** (Lei et al., 2025) adaptively weight document fields or combine planning and graph traversal for structured retrieval, while **AvaTaR** (Wu et al., 2024a) enhances tool-using agents through an explicit optimization loop, it employs a comparator that contrastively reasons over positive and negative examples to iteratively refine agent instructions and prompts. Additionally, it maintains a memory of past failures to guide future decisions, which is why we categorize it as a fine-tuned approach.

Evaluation Metrics. We evaluate methods along two dimensions: retrieval performance and end-to-end performance. For **OTT-QA**, retrieval is measured using Recall@k, which assesses how many relevant passages are retrieved. While we focus only on recall for **STaRK**, we report standard metrics from the original paper: Hit@1, Hit@5, Recall@20, and MRR, capturing both the ranking quality and coverage of retrieved knowledge.

3.1 OTT-QA: Hybrid Text-Table Results

Graph-based retrieval improves Recall@k across all retrieval sizes. Graph-based retrieval consistently improves Recall@k across all retrieval sizes (Figure 4). By adding one-hop neighbors from the chunk-level graph, the number of relevant items retrieved increases at every k, from very small retrieval sets ($k \approx 2$) to larger sets ($k = 100$). This demonstrates that the graph effectively surfaces additional relevant nodes that flat similarity-based retrieval alone cannot capture.

Graph impact is larger for smaller retrieval sets. The impact of the graph is particularly pronounced for smaller retrieval sets. For instance, recall increases by 5.7 points at $k = 20$ compared to 2.8 points at $k = 100$, showing that relative improvements are largest when fewer items are retrieved. This highlights that structural connections in the graph are most beneficial when retrieval is constrained, allowing the system to efficiently identify relevant neighbors that would otherwise be missed.

Effect of metadata on graph building We analyze metadata similarities to guide graph construction. Table-table pairs show strong correlations between title and description similarities (0.846, Table 13), while document-document pairs exhibit notable alignment between topic and content similarities (0.650, Table 14). Cross-modal document-table correlations range from 0.517 to 0.691 (Table 15). These correlations indicate that nodes with high similarity along one metadata dimension are likely to be similar along others, helping form well-connected neighborhoods in the graph. Similarity distributions are mostly bell-shaped with a high-end tail, ensuring broad coverage across the corpus while highlighting strongly related chunks. Most chunks contain only a single entity, which makes entity overlap particularly valuable: two chunks that mention the same entity but discuss different content can still be connected, creating cross-cutting edges that bridge different topics. Percentile-based edge pruning, keeping only the top 5 percent of similarities, preserves these meaningful connections while avoiding overly dense graphs. All detailed distributions, scatter plots, and correlation matrices are provided in Appendix B.

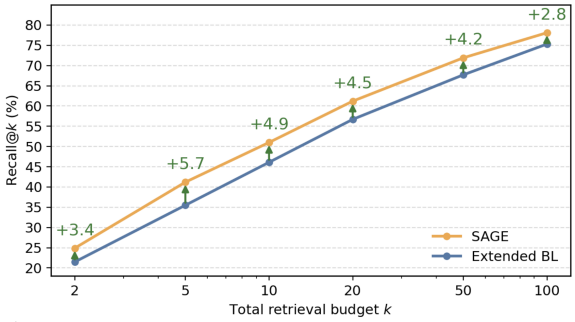

Figure 4: OTT-QA retrieval Recall@k (%, higher is better). BL (Baseline) is a flat hybrid retriever combining BM25 and dense embeddings. BL+Graph retrieves k_1 seeds with BL and adds k_2 one-hop neighbors from the induced graph ($k = k_1 + k_2$), while Extended BL retrieves k items directly with BL. Arrows annotate the absolute gain at each k.

3.2 STaRK: Semi-Structured Data Results

Effect of graph on agentic retriever Similar to our observations on OTT-QA, graph-based retrieval also improves Recall@20 for agentic retrievers across all STaRK datasets (Table 2). By expanding the retrieved set to include connected nodes, the graph surfaces additional relevant nodes that the base agent model misses, leading to higher recall on both Human and Synthetic queries.

Are there dataset-specific patterns where graph augmentation helps more? We observe that graph augmentation provides the largest gains on datasets where nodes contain richer textual information. For example, on AMAZON, which has detailed product metadata per node, Agent+Graph-

Category	Method	AMAZON		MAG		PRIME	
		Human	Synthetic	Human	Synthetic	Human	Synthetic
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 2: Recall@20 performance on Human and Synthetic queries across all STaRK datasets (AMAZON, MAG, PRIME). Highest non-finetuned method is bolded, second highest non-finetuned method is underlined.

Edge achieves an R@20 improvement of +8.49 for Human queries and +4.33 for Synthetic queries. In contrast, datasets with sparser node content, such as MAG and PRIME, show smaller but still measurable improvements. This pattern suggests that chunk-level graphs are particularly effective when nodes carry more context, as richer content allows the graph to surface meaningful multi-hop evidence chains that flat retrieval alone cannot access.

How does this baseline+graph compare vs other state of the art methods SPARK +SAGE-Edge outperforms traditional Dense, Sparse, and Query Expansion baselines by a large margin (Table 2) and achieves performance close to finetuned methods such as AvaTaR, mFAR, and MoR. FocusedR surpasses SPARK +SAGE-Edge on some datasets, reflecting its focus on reasoning over KG triples. This explains why SPARK +SAGE-Edge performs particularly well on AMAZON, where rich node content favors chunk-level graph expansion, while FocusedR has an advantage on MAG and PRIME, where structured graph information is more prominent. Moreover, SPARK +SAGE-Edge is significantly more efficient, requiring only a single LLM call and one retrieval, making it more practical for large-scale deployment.

How efficient is our Agentic approach vs other state of the art methods Table 3 exposes three major efficiency bottlenecks common to existing retrieval methods. First, repeated LLM invocations substantially increase inference cost. Second, iterative retrieval loops introduce execution paths that can grow arbitrarily long. Third, symbolic set-join operations add significant computational overhead. ReAct, Reflexion, and AvaTaR rely on step-by-step retrieval, issuing an LLM call at each action step, which sharply increases both latency and cost as the number of steps grows. In contrast, 4StepFocus and FocusedR perform repeated symbolic pruning with set intersections across multiple passes, leading to substantial runtime growth as query complexity increases. KAR reduces LLM usage to two calls, but still requires multiple dataset-wide retrievals, which limits scalability.

Method	#Retr.	#LLM	#Iter.	Set Join
4StepFocus	1 + n	2	n	Triplet cand.
FocusedR	$\geq$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 3: 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 indicates explicit candidate–neighborhood intersections.

SPARK eliminates all three bottlenecks. It operates with a single LLM call, a single retrieval, and one graph expansion. There are no iterative loops and no symbolic set-join operations. This produces a fixed, lightweight execution path that is dramatically faster while maintaining competitive recall. For real-world deployments where latency and cost are paramount, SPARK delivers a supe-

rior accuracy-per-compute tradeoff. Details about order complexity calculation is in Appendix C.

3.3 Ablation Study

How much of the performance gain is due to the graph structure itself versus the graph expansion strategy? SPARK +SAGE-Edge and SPARK +SAGE +Edge differ in how neighbor expansion is applied for Cypher queries generated by the agent. In SPARK +SAGE-Edge (-Edge), queries with explicit edges do not expand, preserving the semantic relationships between connected nodes. In SPARK +SAGE +Edge (+Edge), neighbors are added regardless of edge semantics.

Both variants improve Recall@20 over the base Agent model (Table 2). The gain is largest for SPARK +SAGE-Edge, where expansion is skipped for edge-containing queries. This is because preserving the semantic relationships in the graph avoids adding less relevant neighbors, maintaining high-quality candidates. At the same time, applying graph expansion for queries without edges brings in additional relevant nodes. Together, these results show that both the underlying graph structure and the expansion strategy are equally important for improving retrieval.

	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
Agent	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 4: 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..

What is the effect of the initial retriever on the graph-based expansion? The effectiveness of graph augmentation depends strongly on the quality of the initial retriever, as shown in Table 4. For the dense and sparse hybrid baseline, Recall@20 is substantially lower than that of the agent retriever, and the improvement from graph expansion is limited. In some cases, particularly on the PRIME and MAG subsets at low retrieval depths, the delta is negligible or even negative, for example, Hybrid Human MAG goes from 46.6 to 46.4. This is partly because the hybrid retriever does not take node type into account, while graph expansion tends to favor nodes with embeddings similar to the seed nodes, which are often of the same type. Without a strong initial candidate set and guidance on node type, the expansion can introduce irrelevant nodes, limiting gains.

By contrast, the agent retriever provides a higher initial Recall@20 and determines the node type. Graph expansion with the agent consistently improves performance across all datasets and query types, for example, AMAZON Human goes from 33.6 to 42.1. The combination of stronger initial candidates and type-aware retrieval allows the graph to add relevant nodes, amplifying the benefits of expansion. Full Recall@k curves showing these trends are provided in Appendix D.

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 5: Error distribution (%) across STaRK datasets.

Constraint extraction failures dominate across AMAZON and PRIME, reflecting challenges in translating complex product attributes or biomedical predicates into precise graph queries. MAG is primarily affected by semantic mismatches, due to dense technical abstracts where terminology and phrasing differ between queries and graph entities. Over-retrieval occurs in all datasets when filters are insufficient. PRIME uniquely exhibits substan-

tial graph structure errors, consistent with its richer schema requiring correct edge and node selection. Data errors are generally minor but still contribute small percentages, particularly in MAG where gold annotations are incomplete (Table 5).

Overall, these patterns highlight a trade-off between *query translation*, *data quality*, and *semantic alignment*. AMAZON queries fail due to diverse terminology and units, MAG queries due to complex content that is poorly captured by structured graph traversal, and PRIME queries due to navigating its rich schema.

4 Related Work

Retrieval-augmented QA over heterogeneous corpora must handle not only textual relevance but also *structure*: tables have schema and join constraints, documents have latent topical links, and many real collections expose explicit relations. Below, we summarize different techniques:

Table-centric retrieval and dataset discovery For table repositories, a major theme is retrieving *related* tables using structural criteria such as unionability and joinability. Starmie (Fan et al., 2022), DeepJoin (Dong et al., 2022), and WarpGate (Cong et al., 2022) learn or encode column/table representations for efficient join or union search, typically accelerated via ANN indexing. Other work targets specialized relational augmentation signals, e.g., correlated dataset search via sketch-based indexing (Santos et al., 2022). For end-to-end table QA, systems improve practical retrieval quality with cascaded pipelines that combine sparse filtering, dense retrieval, and reranking (Singh et al., 2025), while LLM-based dense retrieval techniques (e.g., HyDE) can improve zero-shot retrieval without labeled relevance data.

Graph-based indexing for retrieval and RAG Graph-based RAG methods explicitly organize information into graphs to retrieve connected context or aggregate evidence. Microsoft GraphRAG (Edge et al., 2024b) builds an entity graph and community summaries for corpus-level questions; GRAG (Peng et al., 2024) focuses on retrieving textual subgraphs and incorporating topology into generation; and PathRAG (Chen et al., 2025a) emphasizes pruning redundancy and path-based prompting. In multi-document QA, knowledge-graph prompting constructs passage graphs and uses traversal to assemble supporting context (Wang et al., 2023). A broader survey (Han et al., 2024) highlights that graph form and domain constraints strongly shape the design space.

LM+KG reasoning and agentic retrieval with tools When explicit KGs are available, models such as QA-GNN (Wang et al., 2024; Sun et al., 2018, 2019; Yasunaga et al., 2021; Pan et al., 2024) combine LM-based relevance estimation with graph neural reasoning. In parallel, agentic frameworks treat retrieval as a plan–execute process over external tools: ReAct interleaves reasoning and actions, Reflexion improves agents via feedback and self-reflection, AvaTaR optimizes tool use via contrastive reasoning.

Prior work often targets a single structure type (table repositories, explicit KGs, or graph indices for text corpora). Our framework is retrieval-centric and structure-aware: we build chunk graphs offline when relations are implicit, preserve native relations when graphs are explicit, and use a simple *seed* $\rightarrow$ *expand* $\rightarrow$ *rerank* operator to select evidence under a fixed budget.

5 Conclusion

We present SAGE, a structure-aware graph retrieval framework that adapts retrieval operators to data topology by separating offline graph construction from online query-conditioned traversal. Metadata-driven similarity and pruning build graphs for implicit corpora, while native schema edges are preserved for explicit KGs, with operators selected at query time based on graph density and schema availability.

On OTT-QA, similarity-driven expansion boosts Recall@k by 2.8–5.7 points over strong hybrid baselines, retrieving more coherent multi-hop evidence with compact context. On STaRK, agentic retrieval with selective expansion approaches fine-tuned performance without training, using symbolic Cypher queries to navigate dense graphs while limiting neighbor noise. Overall, structure-aware retrieval that leverages graph connectivity can match or outperform strong hybrid baselines without task-specific training.

Future work may include instruction tuning or fine-tuning the agent to generate more accurate and efficient Cypher queries, as well as extending the framework to multimodal graphs that incorporate images and other media through cross-modal links.

Limitations

Our study focuses on two representative regimes and relies on LLM-based agents for STaRK, which introduces latency and computational cost compared to traditional retrieval. This tradeoff enables training-free symbolic Cypher generation that provides precise schema-aware traversal while avoiding neighbor noise in dense graphs. The framework depends on initial seed quality, as graph expansion amplifies rather than replaces base retrieval. Error analysis reveals constraint extraction and filtering (30 to 36 percent of errors) as the dominant failure mode, reflecting the difficulty of mapping informal queries to structured predicates without task-specific training. Despite these limitations, we excel at surfacing structurally connected multi-hop evidence that flat retrieval misses, achieving 2.8 to 5.7 point gains on OTT-QA and approaching fine-tuned baselines on STaRK without training, with particularly strong performance on complex queries where graph topology provides discriminative structure.

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 Yang Wang, and William W. Cohen. 2021. Open question answering over tables and text. In *International Conference on Learning Representations*.

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*.

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. 2024. An enhanced prompt-based llm reasoning scheme via knowledge graph-integrated collaboration. In *International Conference on Artificial Neural Networks*, pages 251–265.

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*.

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.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3580–3599.

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.

Aécio 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*.

Haitian Sun, Tania Bedrax-Weiss, and William Cohen. 2019. Pullnet: Open domain question answering with iterative retrieval on knowledge bases and text. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 2380–2390.

Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William Cohen. 2018. Open domain question answering using early fusion of knowledge bases and text. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 4231–4242.

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.

Yu Wang, Nedim Lipka, Ryan A Rossi, Alexa Siu, Ruiyi Zhang, and Tyler Derr. 2024. Knowledge graph prompting for multi-document question answering. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 19206–19214.

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*.

A Dataset Statistics

A.1 OTT-QA

OTT-QA consists of a heterogeneous graph connecting document and table chunks (Table 6). 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 6: OTT-QA dataset statistics showing node types, edge types, and their counts.

A.2 STaRK

A.2.1 AMAZON

The AMAZON graph is dominated by product reviews (Tables 7 and 8), resulting in extremely dense connectivity between products and user-generated text. This rich textual content and high edge density make AMAZON particularly well-suited for graph-based expansion and semantic propagation.

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

Table 7: AMAZON node type distribution from STaRK dataset.

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 8: AMAZON edge type distribution from STaRK dataset.

A.2.2 MAG

MAG represents a large scholarly graph with diverse entity types but comparatively limited textual richness per node (Tables 9 and 10). 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 9: 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 10: MAG edge type distribution from STaRK dataset.

A.2.3 PRIME

PRIME is a highly structured biomedical graph with many fine-grained entity and relation types (Tables 11 and 12). 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.

B OTTQA graph building statistics

B.1 Table–Table Similarity Analysis

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

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 11: 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 12: PRIME edge type distribution from STaRK dataset.

Column similarity and entity distribution Figure 5 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.

Description and title similarity Figure 6 show 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.

Column–title, column–description, title–description scatter plots Figure 7 show scatter plots of column–title, column–description, and title–description similarities, respectively. Strong diagonal trends indicate that similar

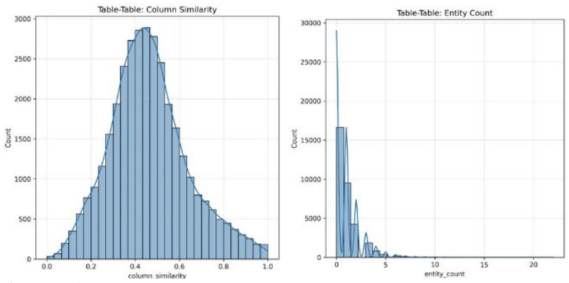

Figure 5: Left: column similarity distribution. Right: number of entities per table chunk (left-skewed).

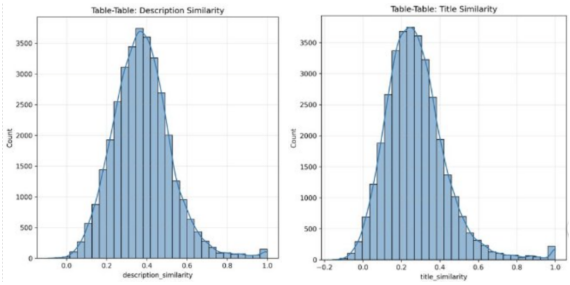

Figure 6: Left: description similarity. Right: title similarity. Both distributions are similar, with a slight rise at high similarity.

columns tend to co-occur with similar titles and descriptions.

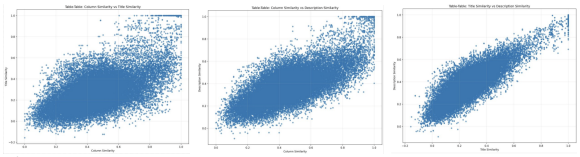

Figure 7: Scatter plots of column–title (left), column–description (center), and title–description (right) similarities. Diagonal trends indicate strong alignment.

Correlation matrix Table 13 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. This confirms that LLM-based summarization is unnecessary, and BM25 weights suffice.

B.2 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 8 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.

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

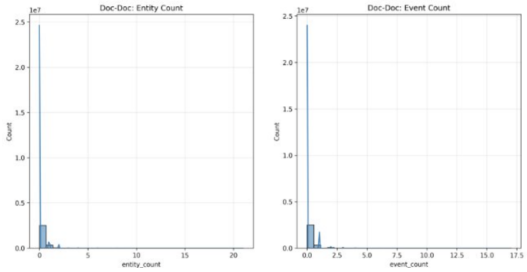

Figure 8: Distribution of entities and events per document chunk. Most chunks contain only one entity, reducing the need for LLM-based extraction.

Topic and content similarity distributions Figure 9 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.

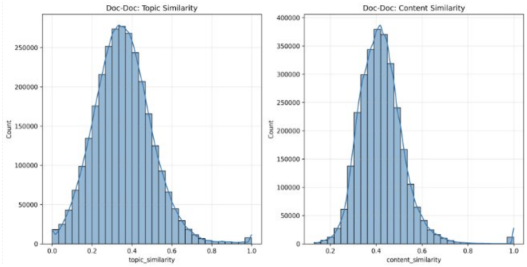

Figure 9: Histograms of topic similarity (left) and content similarity (right) between document chunks. Bell-shaped distributions with a high-similarity spike are observed.

Topic–content scatter plot Figure 10 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.

Correlation matrix Table 14 shows the correlation matrix of topic similarity, content similarity, and entity/event counts. Strong correlations confirm that document chunks are semantically aligned, and simple similarity measures suffice for graph construction without LLMs.

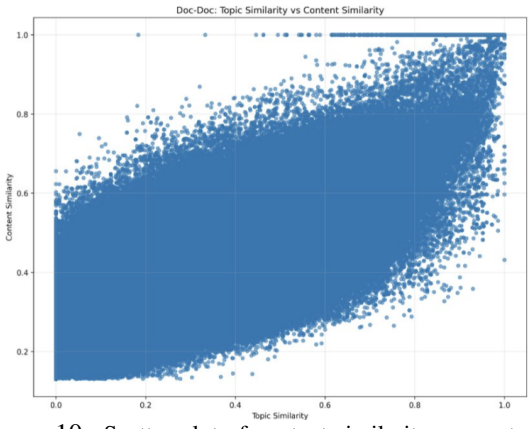

Figure 10: Scatter plot of content similarity versus topic similarity. Diagonal concentration indicates strong alignment between the two metrics.

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

B.3 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 11 show 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.

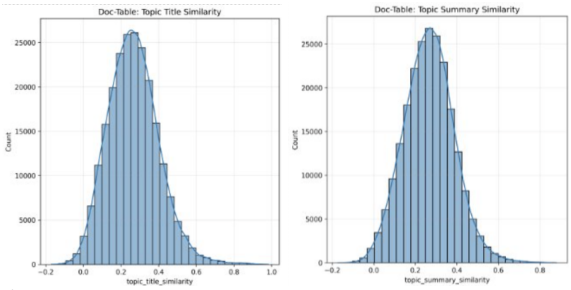

Figure 11: Left: topic–title similarity distribution. Right: topic–summary similarity distribution. Both are bell-shaped.

Column similarity and entity counts Figures 12 and 12 show 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.

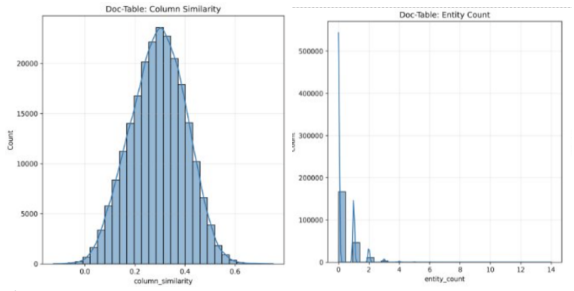

Figure 12: Left: column similarity distribution. Right: entity count per table chunk (left-skewed).

Scatter plots of topic, title, summary, and column similarities Figures 13, 13, and 13 show 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.

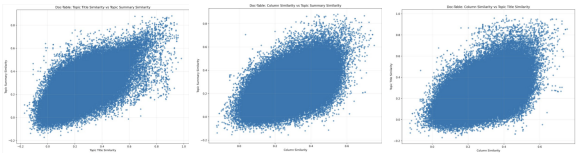

Figure 13: Scatter plots showing alignment between document and table chunks: topic–title vs topic–summary (left), column vs topic–summary (center), column vs topic–title (right). Diagonal trends indicate strong alignment.

Correlation matrix Table 15 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 15: Doc-Table Correlation Matrix. Abbreviations: (1) column_similarity, (2) entity_count, (3) topic_title_similarity, and (4) topic_summary_similarity

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

C.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:

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}

Runtime Analysis:

- **Step 1:** $O(1)$ — single LLM call.
- **Step 2:** $O(n \cdot |T| \cdot \text{deg})$ — 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:

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

C.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:

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:

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**
    $S(h_i) \leftarrow S(h_i) \cap$ neighbors$(h_i, e_i, -)$
    $S(t_i) \leftarrow S(t_i) \cap$ neighbors$(t_i, e_i, +)$
  end
until *no changes in any* $S(x)$
return S

Runtime Analysis:

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

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

C.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.

C.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.

C.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.

C.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 → retrieval → 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.

C.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.

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

D.1 AMAZON

Figures 14 and 15 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.

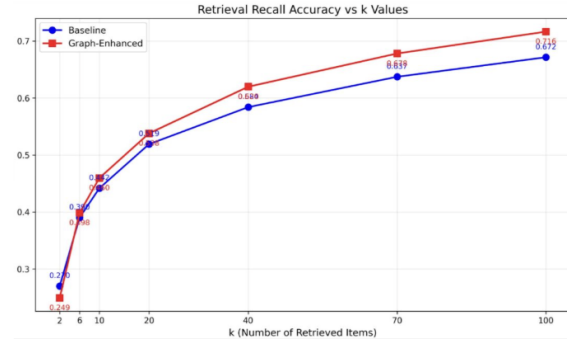

Figure 14: Recall@k for Synthetic queries on AMAZON. Graph expansion improves recall across all k.

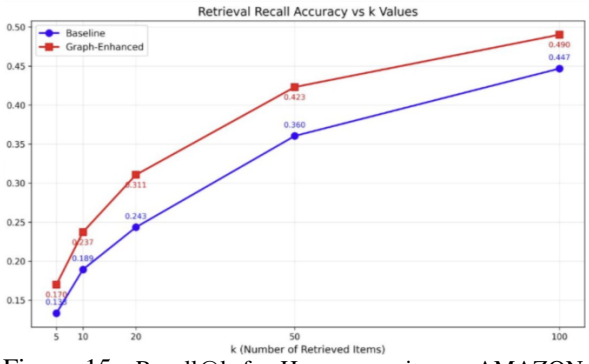

Figure 15: Recall@k for Human queries on AMAZON. Graph expansion consistently increases recall.

D.2 MAG

Figures 16 and 17 show recall@k for MAG. Here, graph expansion helps primarily at higher k values ($k10$), 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.

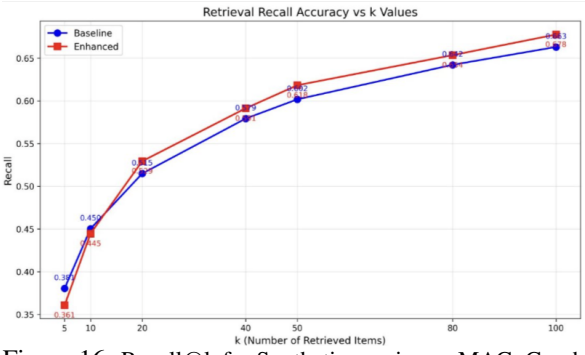

Figure 16: Recall@k for Synthetic queries on MAG. Graph expansion improves recall at higher k, but hurts at low k.

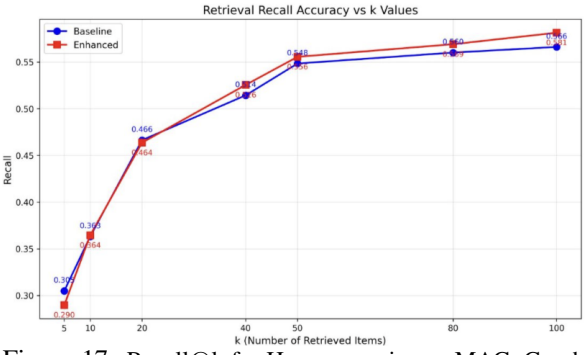

Figure 17: Recall@k for Human queries on MAG. Graph expansion shows gains only at larger k.

D.3 PRIME

Figures 18 and 19 show recall@k for PRIME. Similar to MAG, graph expansion benefits are observed primarily at higher k values ($k20$), 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.

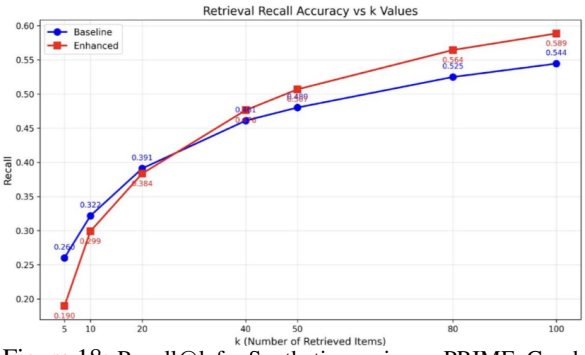

Figure 18: Recall@k for Synthetic queries on PRIME. Graph expansion is effective only at higher k.

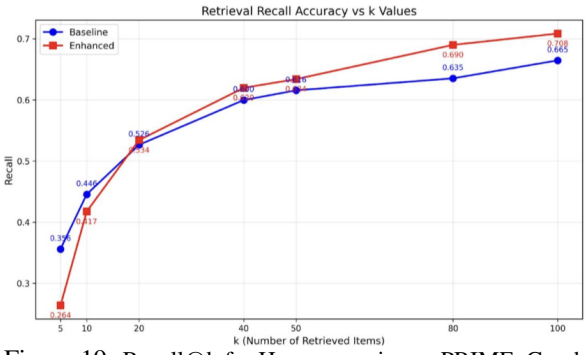

Figure 19: Recall@k for Human queries on PRIME. Graph expansion helps mainly at larger k.

E Additional Metrics for STaRK

Table 16 presents Hits@1, Hits@5, and MRR scores for all methods across AMAZON, MAG, and PRIME datasets.

F 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.

G Prompts for Agentic baseline

G.1 AMAZON

Listing 2: 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'
```

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

```
  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
```

G.2 MAG

Listing 3: 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
```

G.3 PRIME

Listing 4: 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
    interactions
(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.detailsJson (str)
    Contains: summary, aliases,
    location

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

Drug:
```

```
- drug.nodeId (int, unique)
- drug.name (str)
- drug.sourceId (str)
- drug.detailsJson (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.detailsJson (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
```