Autonomous Knowledge Graph Exploration with Adaptive Breadth-Depth Retrieval

Anonymous ACL submission

Abstract

Retrieving evidence for language model queries from knowledge graphs requires balancing broad search across the graph with targeted multi-hop traversal to follow relational links. Similarity-based retrievers provide coverage but remain shallow, whereas traversal-based methods rely on selecting seed nodes to start exploration, which can fail when queries span multiple entities and relations. We introduce **DGR**: DYNAMIC GRAPH RETRIEVER, an agentic KG retriever that gives a language model control over this breadth-depth trade-off using a two-operation toolset: global lexical search over node descriptors and one-hop neighborhood exploration that composes into multi-hop traversal. DGR alternates between breadth-oriented discovery and depth-oriented expansion without depending on a fragile seed selection, a pre-set hop depth, or graph training. DGR autonomously adapts tool use to queries, using global search for language-heavy queries and neighborhood exploration for relation-heavy queries. On STaRK, DGR reaches 59.1% average Hit@1, improving Hit@1 by up to 9.5% and MRR by up to 7.5% over trained retrievers and agentic baselines while maintaining performance on various graph regimes. Finally, we distill DGR's tool-use trajectories from a large teacher into an 8B model via label-free imitation, improving Hit@1 by 7.0%, 26.6%, and 13.5% over the base 8B model on AMAZON, MAG, and PRIME datasets, and retaining up to 98.5% of the teacher's Hit@1 rate.

1 Introduction

Knowledge-intensive NLP systems rely on retrieval to ground language model outputs in external evidence, from retrieval-augmented generation (RAG) to systems and memory modules that operate over *semi-structured* knowledge bases (SKB) that mix text with relational information (Lewis et al., 2020; Guu et al., 2020; Karpukhin et al., 2020; Izacard and Grave, 2021; Mavromatis and Karypis, 2025; Chen et al., 2025). Knowledge graphs (KGs) are a natural data representation for this setting because they organize evidence around entities and typed edges, support reuse across queries, and enforce relational constraints that a flat text index cannot express. This has motivated graph-aware retrievers and graph-grounded generation methods, including graph-based RAG and SKB retrievers that combine text and relational data (Edge et al., 2025; Zhu et al., 2025).

Retrieving evidence from KGs is challenging because it requires coordinating two competing search modes (Wu et al., 2024b; Lee et al., 2025). Many queries require *breadth*: they mention multiple entities or loosely connected concepts, so the retriever must cover the graph broadly to reach the right region. Other queries require *depth*: the supporting evidence only appears after following specific multi-hop relational paths. Similarity-based retrievers provide global coverage but often remain shallow and underuse relational structure, whereas traversal-based methods can be brittle because they must choose seed entities to start exploration; when seeds are incomplete or ambiguous, the search stays local and misses evidence elsewhere.

Existing work tackles these requirements in isolation rather than jointly. *Structure-aware* retrievers extend text-based retrieval with relational structure, for example, by learning node embeddings that aggregate information from nearby neighbors or by generating candidates using a local graph neighborhood before ranking them (Lee et al., 2025; Zhu et al., 2025; Lei et al., 2025). These methods capture local structure, but they typically encode a fixed neighborhood around each node, so deeper multi-hop queries require expanding the encoded context or stacking additional message-passing and retrieval stages, which increases complexity and cost. By contrast, *traversal-based ap-*

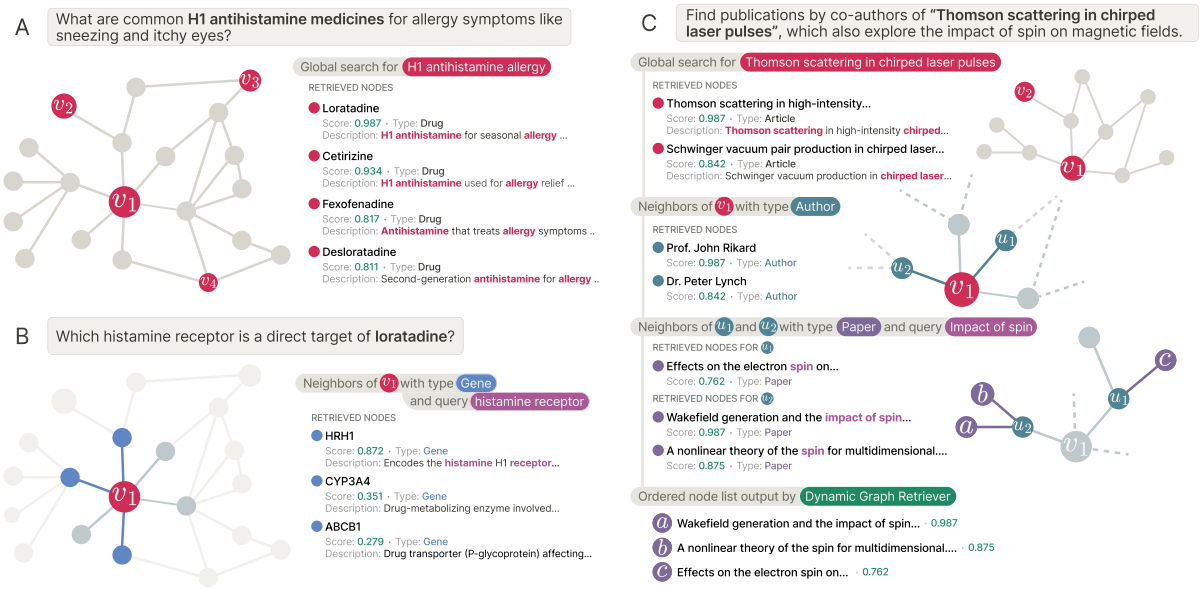

Figure 1: Overview of DYNAMIC GRAPH RETRIEVER. DGR interacts with a KG through a minimal two-tool interface: **(a)** For text-dominant queries, DGR emphasizes breadth by issuing GLOBAL SEARCH to retrieve a broad set of candidates. **(b)** For relation-focused queries, DGR applies NEIGHBORHOOD EXPLORATION starting from a previously retrieved node (in this case, a drug) and expanding to related entities, enabling targeted relational retrieval. **(c)** For relation-dominant queries, DGR performs multi-hop retrieval by alternating GLOBAL SEARCH and NEIGHBORHOOD EXPLORATION: it retrieves an initial node (e.g., a paper), expands to related entities (e.g., co-authors), and continues expanding and filtering (e.g., papers connected to each author that match query keywords) to recover an ordered set of relevant evidence.

proaches perform multi-hop exploration, but they depend on identifying a small set of seed entities from which exploration begins (Markowitz et al., 2024; Sun et al., 2024). When seeds are incomplete or ambiguous, exploration stays local and misses relevant information elsewhere in the graph. Many systems also rely on task- or graph-specific training to learn traversal or scoring policies, which limits transfer across domains and graph schemas (Li et al., 2025a; Lei et al., 2025; Wu et al., 2024a; Yu et al., 2025). As a result, existing methods struggle to combine global search with targeted relational reasoning for adaptive retrieval.

Present work. We introduce **DGR**: DYNAMIC GRAPH RETRIEVER, an agentic KG retriever that gives a language model control over evidence discovery using a minimal set of tools for global lexical search and neighborhood exploration (Yao et al., 2023; Schick et al., 2023). DGR does not require selecting seed entities to start exploration or establishing a maximum hop depth in advance; instead, the model alternates between broad global search and targeted multi-hop traversal based on the query and what it has retrieved so far. We evaluate DGR on the heterogeneous graphs in the STaRK retrieval benchmark and show consistent gains across all settings. We further study compute-accuracy tradeoffs by varying the tool-call budget and the number of parallel agents, and we distill DGR's tool-use policy into an 8B model via label-free trajectory imitation, preserving most of the performance of teacher model at substantially lower inference cost (Kang et al., 2025).

Our contributions are threefold. (i) We introduce DYNAMIC GRAPH RETRIEVER, a training-free retrieval framework that equips language models with a minimal but expressive tool interface for adaptive retrieval from KGs. (ii) We show that DGR balances breadth-oriented retrieval with depth-oriented multi-hop traversal without task- or graph-specific training, achieving strong performance on STaRK. (iii) We distill DGR's tool-use policy without labeled supervision into a compact Qwen3-8B model (Yang et al., 2025a; Kang et al., 2025), preserving retrieval quality while reducing inference cost.

2 Related Work

Knowledge graphs for document-centric RAG. Retrieval-Augmented Generation (RAG) grounds

LLM outputs in external evidence by retrieving relevant context from a corpus or index (Lewis et al., 2020; Guu et al., 2020). Recent work injects structure by building graphs over textual units and using connectivity to aggregate evidence. GraphRAG performs local-to-global retrieval over an entity-centric graph (Edge et al., 2025), while KG-guided methods steer evidence selection using external relations (Zhu et al., 2025). Related approaches retrieve textual subgraphs to support multi-hop queries under context limits (Hu et al., 2025; He et al., 2024; Li et al., 2025b). These approaches primarily focus on improving how textual evidence is organized or aggregated, but the retrieval process itself is typically static.

Retrieval over semi-structured knowledge bases. Complementary to document-centric graph indices, semi-structured knowledge base (SKB) retrievers directly combine text and explicit relations. KAR grounds query expansion in KG structure (Xia et al., 2025), and hybrid systems mix graph and text channels with iterative refinement, e.g., Graph-Search and HybGRAG (Yang et al., 2025b; Lee et al., 2025); CoRAG highlights cooperative hybrid retrieval that preserves global semantic access beyond local neighborhoods (Zheng et al., 2025). In parallel, parametric retrievers such as MoR and mFAR learn to fuse lexical, semantic, and structural signals for ranking (Lei et al., 2025; Li et al., 2025a). Across these variants, retrieval is framed as scoring candidates from a static index. DGR differs in that retrieval is formulated as an interactive process: the model dynamically switches between global search and neighborhood expansion, guided by the query requirements and without relying on task-specific supervision.

Agents for multi-hop KG retrieval and QA. A separate line of work treats the KG as an environment for iterative interaction. Earlier agents follow relation paths using reinforcement learning or learned policies (Das et al., 2018; Xiong et al., 2017; Sun et al., 2019; Asai et al., 2020). In the LLM era, tool-use frameworks such as ReAct (Yao et al., 2023) and prompt-optimization methods such as AvaTaR (Wu et al., 2024a) enable interactive evidence gathering. Within KG retrieval, recent traversal-based approaches expand from seed entities using prompted heuristics or learned policies, including Tree-of-Traversals, Think-on-Graph, and GraphFlow (Markowitz et al., 2024; Sun et al., 2024; Yu et al., 2025); related KG-grounded reasoning methods also emphasize multi-step planning or navigation on the KG (Luo et al., 2024; Sun et al., 2025). Traversal-based agents are effective when the correct starting entities are known, but they are prone to anchoring errors and can over-commit to local neighborhoods once exploration begins. In DGR, global search remains available throughout the trajectory, allowing the agent to retain a complete view of the KG at each step. This design enables coordination between global discovery and deep multi-hop expansion within a single retrieval process.

Existing work varies in whether it treats retrieval as static ranking over an index or as sequential decision-making on the graph, and in whether it requires graph-specific supervision to learn a ranking function or a traversal policy. DGR adapts its search strategy online through a minimal, graph-native tool interface. It is training-free; when needed, its tool-use policy can be distilled into compact models from interaction trajectories without ground-truth relevance labels, improving efficiency while preserving retrieval quality.

3 Dynamic Graph Retriever

We study retrieval over a knowledge graph $G = \langle V, E, \phi_V, \phi_E, d_V \rangle$, where V and E denote entities and edges, ϕ_V and ϕ_E assign a type to each node and relation, and $d_V(v)$ denotes the text attributes associated with node v, such as titles, descriptions, or other metadata. Given a natural-language query Q, retrieval is formulated as an interactive process in which an agent $\mathcal{A} = \langle \text{LLM}, \mathcal{T} \rangle$ queries the graph through a tool interface $\mathcal{T}$ (Yao et al., 2023; Schick et al., 2023) and produces a trajectory $\tau = \big((s_1, A_1, o_1), \ldots, (s_T, A_T, o_T)\big)$. At step t, s_t contains Q and the interaction history, A_t is a sequence of tool invocations, and o_t is the observation returned after executing A_t.

Throughout the trajectory, the agent maintains an ordered list of retrieved nodes $\mathcal{R}$. At each step, it can SELECT nodes returned by tools and append them to $\mathcal{R}$, or terminate by issuing a dedicated FINISH action. Execution ends either when the agent calls FINISH or when the maximum trajectory length $T_{\max}$ is reached. The final retrieval output is the ranked list $\mathcal{R} = (v_1, v_2, \ldots)$, where earlier selections receive higher rank.

To rank candidate nodes returned by tools, we use a relevance function $\text{rel}(q, d_V(v)) \in \mathbb{R}_{\geq 0}$ that scores node v for a textual subquery q provided by the agent as a tool parameter. We implement

rel with BM25 (Robertson and Zaragoza, 2009) over an inverted index of node textual attributes (Manning et al., 2008), yielding fast and stable scoring for the many short, evolving subqueries issued during exploration.

3.1 Tools

We implement the interaction described above through a small set of retrieval tools. Each tool returns a candidate set of nodes, which the agent may append to $\mathcal{R}$ or use to guide subsequent steps.

Global search retrieves the k highest-scoring nodes in the graph under rel for an agent-issued subquery q, as shown in Figure 1a:

$$\text{Search}_G(q, k) \;:=\; \underset{v \in V}{\text{Top-k}}\ \text{rel}(q, d_V(v))$$

This tool provides broad entry points into the graph and is primarily used (i) to locate entities related to the user query Q, which will then be used for further exploration, and (ii) to handle cases where direct text matching suffices without requiring multi-hop reasoning.

Neighborhood exploration returns adjacent nodes of a node v filtered by optional node and edge type constraints $F := (F_V, F_E)$ selected by the agent as tool parameters, and optionally ranked using an agent-generated subquery q (Figure 1b).

The filtered one-hop neighborhood N_F of a node v is defined as:

$$N_F(v) := \left\{ u \in N(v) \;\middle|\; \begin{array}{l} \phi_V(u) \in F_V, \\ \phi_E(\{u, v\}) \in F_E \end{array} \right\}$$

where $N(v)$ denotes the open neighborhood of v and $\{u, v\}$ denotes the edge connecting v and u, regardless of direction. Edge directionality and relation types are exposed in the tool output. To control the size of the retrieved neighborhood, we introduce a fixed retrieval budget $k \in \mathbb{N}$:

$$\text{Neighbors}(v, q, F) := \underset{u \in N_F(v)}{\text{Top-k}}\ \text{rel}(q, d_V(u))$$

We restrict Neighbors to single-hop expansion so that multi-hop exploration emerges through composition rather than fixed-depth traversal (Figure 1c).

3.2 Parallel Exploration

We increase robustness by running n independent instances of the same agent in parallel and aggregating their retrieved lists, akin to self-consistency and voting-based ensembling in LLM reasoning (Wang et al., 2023; Kaesberg et al., 2025). Each agent produces an ordered list of retrieved nodes $\mathcal{R}^{(i)} = (v_1^{(i)}, v_2^{(i)}, \dots)$ from an independent trajectory. We then combine these lists using a simple rank-fusion rule inspired by classical rank aggregation and data fusion methods (Fagin et al., 2003; Cormack et al., 2009).

Concretely, we concatenate the per-agent lists in agent order to form:

$$L \;:=\; \mathcal{R}^{(1)} \,\|\, \mathcal{R}^{(2)} \,\|\, \cdots \,\|\, \mathcal{R}^{(n)},$$

and let $\mathcal{V}_L$ be the set of unique nodes in L. The final ranking $\mathcal{R}$ orders nodes by decreasing frequency in L (vote count), breaking ties by the earliest position at which a node appears in any trajectory, favoring nodes discovered earlier during exploration.

3.3 Agent Distillation

While DGR operates on off-the-shelf models, its behavior can be distilled into a smaller language model to reduce inference cost and latency (Hinton et al., 2015). We adopt a standard teacher–student paradigm in which a student model imitates the tool-usage trajectories of a stronger teacher LLM via supervised fine-tuning (Schick et al., 2023).

Trajectory generation. For each training query Q on a given graph G, we run the teacher agent to collect trajectories τ as defined in Section 3. Each trajectory contains the full interaction record: the agent's tool calls and parameters interleaved with the resulting tool observations.

Training objective. The student is trained with next-token prediction on the collected trajectories (Ouyang et al., 2022). We compute loss only on assistant-authored tokens; user messages and tool outputs are masked (Huerta-Enochian and Ko, 2024; Shi et al., 2024). This trains the student to reproduce the teacher's decisions, which tools to invoke and how to parameterize them, while tool execution remains external to the model.

Label-free supervision. Importantly, distillation does not require ground-truth evidence nodes for queries. Supervision is derived solely from teacher trajectories, making the approach applicable in realistic settings where relevance labels are unavailable: one can run a strong teacher to generate trajectories on a target graph and then fine-tune a smaller model directly from these interactions (Schick et al., 2023; Kang et al., 2025).

Category	Method	AMAZON Hit@1	AMAZON R@20	AMAZON MRR	MAG Hit@1	MAG R@20	MAG MRR	PRIME Hit@1	PRIME R@20	PRIME MRR	Average Hit@1	Average R@20	Average MRR
Training-free	*Retrieval-based*												
	BM25	44.94	53.77	55.30	25.85	45.69	34.91	12.75	31.25	19.84	27.85	43.57	36.68
	Dense	39.16	53.29	50.35	29.08	48.36	38.62	12.63	36.00	21.41	26.96	45.88	36.79
	KAR	54.20	57.24	61.29	50.47	60.28	57.51	30.35	50.81	39.22	45.01	56.11	52.67
	Agent-based												
	Think-on-Graph + 4o	20.67	25.81	30.90	23.33	48.03	36.38	16.67	54.35	27.02	20.22	42.73	31.43
	Think-on-Graph + LLaMA3	4.21	2.61	5.25	12.00	6.77	12.67	21.92	33.84	26.61	12.71	14.41	14.84
	DGR	**55.82**	60.61	**64.77**	**73.40**	**84.47**	**79.87**	48.20	69.46	57.68	**59.14**	**71.51**	**67.44**
	DGR + 4o	55.13	57.18	64.29	67.01	79.79	75.46	36.01	60.13	46.44	52.72	65.70	62.06
Requires training on target graph	*Retrieval-based*												
	mFAR	53.0	**66.3**	64.3	55.9	74.1	64.3	40.0	72.6	52.0	49.63	71.00	60.20
	MoR	52.19	59.92	62.24	58.19	75.01	67.14	36.41	63.48	46.92	48.93	66.14	58.77
	Agent-based												
	GraphFlow	47.85	36.15	55.49	39.09	57.18	47.82	**51.39**	**79.71**	**61.37**	46.11	57.68	54.89
	AvaTaR	49.90	60.60	58.70	44.36	50.63	51.15	18.40	39.30	26.73	37.55	50.18	45.53
	DGR distilled	54.99	60.31	64.24	61.66	81.39	70.09	31.87	57.22	41.08	49.51	66.31	58.47

Table 1: Retrieval performance on STaRK synthetic test sets. Dark green and light green indicate best and second-best in the training-free category, respectively. Dark blue and light blue indicate best and second-best in the requires-training category, respectively. **Bold** indicates the best result overall for each metric column.

4 Experimental Setup

We measure retrieval performance on STaRK, a benchmark for entity-level retrieval over heterogeneous, text-rich KGs (Wu et al., 2024b).

4.1 Benchmark

STaRK comprises three large, heterogeneous knowledge graphs. **AMAZON** is an e-commerce graph with roughly 1M entities and 9.4M relations, constructed from Amazon metadata, reviews (He and McAuley, 2016), and question–answer pairs (McAuley et al., 2015). **MAG** is a scholarly graph with 1.9M entities and 39.8M relations derived from the Microsoft Academic Graph (Wang et al., 2020). **PRIME** is a biomedical graph built from PrimeKG (Chandak et al., 2023), containing 129K entities and 8.1M relations. Each node is associated with text-rich attributes, making STaRK a natural testbed for hybrid retrieval over structured and textual signals. Given a query, the retriever must return a ranked list of nodes that support the answer. We report the agent configuration and hyperparameters in Appendix A.2.

4.2 Baselines and metrics

We compare with representative retrieval-based and agent-based baselines, focusing on methods that report results on all three graphs, as our goal is to evaluate performance consistently across different regimes and assess generality.

Retrieval-based. BM25 (Robertson and Zaragoza, 2009) is the same sparse, lexical retriever used for global search. We also include dense embedding retrievers that rank nodes by cosine similarity, using **ada-002** and **GritLM-7B**, an instruction-tuned 7B encoder (Muennighoff et al., 2025). **mFAR** (Li et al., 2025a) is a multi-field adaptive retriever that combines keyword matching with embedding similarity to learn query-dependent weights over different node fields. **KAR** (Xia et al., 2025) augments queries with knowledge-aware expansions and applies relation-type constraints during retrieval. **MoR** (Lei et al., 2025) is a trained retriever that combines multiple retrieval objectives.

Agent-based. Think-on-Graph (Sun et al., 2024) is a training-free LLM agent that iteratively expands paths in the graph using beam search. **GraphFlow** (Yu et al., 2025) learns a policy for generating multi-hop retrieval trajectories using GFlowNets (Bengio et al., 2021). **AvaTaR** (Wu et al., 2024a) is a tool-using agent that optimizes prompting from positive and negative trajectories.

Results for KAR, mFAR, MoR, AvaTaR, and GraphFlow are reported as in their respective papers, which evaluate on the official STaRK splits and metrics. For Think-on-Graph, we report the numbers provided in the GraphFlow study, which includes a direct comparison to Think-on-Graph under the same STaRK setup (Yu et al., 2025; Sun et al., 2024).

Metrics. We follow the STaRK protocol and report Hit@1, Hit@5, Recall@20 (R@20), and Mean Reciprocal Rank (MRR), which capture top-rank precision, coverage of the ground-truth set, and overall ranking quality. Note that Hit@5 is reported in Table 5 in the Appendix.

4.3 Distillation Setup

For each graph, we collect teacher trajectories on the training split to distill DGR into a smaller, lower-cost model (Section 3.3), offering a viable alternative when under tighter compute budgets. Using GPT-4.1 as the teacher, we run DGR three times per training query with stochastic decoding (temperature $= 0.7$), producing three trajectories per query. We cap the distillation budget by subsampling up to 6,000 training queries per graph, yielding at most 18,000 trajectories per graph (full statistics in Table 3), summingå to 94.4 million tokens. Each trajectory is limited to $T_{\max}$=20 steps and ends when the agent issues FINISH or reaches the step limit. We apply no trajectory filtering or rejection sampling, preserving a label-free setting. We then distill a Qwen3-8B (Yang et al., 2025a) student via supervised fine-tuning with LoRA adapters (Hu et al., 2021), using next-token prediction over assistant-authored tokens only. We train for one epoch with a 16,384-token context length using AdamW (Loshchilov and Hutter, 2019) at learning rate 1×10^{-5}, selecting checkpoints via early stopping on the official validation split. Training runs on a single NVIDIA H100 GPU and completes in approximately five hours.

5 Results

5.1 Benchmarking

Table 1 reports retrieval performance on STaRK, grouped by training regime. Across all methods assessed, DGR achieves the best average performance.

Classical retrievers remain strong baselines on AMAZON, when queries are predominantly descriptive. By incorporating local structural cues, KAR improves over lexical methods, but its shallow neighborhood expansion is limited on multi-hop queries (Xia et al., 2025).

Think-on-Graph and GraphFlow highlight the benefits of multi-hop traversal, performing well on PRIME. Think-on-Graph is appealing due to its training-free setup, and GraphFlow shows that strong performance can be achieved with smaller backbones through reinforcement learning. However, both degrade on AMAZON's text-heavy, broad queries, as they lack global search primitives and can be sensitive to brittle anchoring and entity identification (Sun et al., 2024; Yu et al., 2025).

DGR performs consistently across regimes and is especially strong on MAG. This pattern aligns with its tool design. Global search offers a reliable anchor for text-heavy queries and supports strong top-rank accuracy on AMAZON. Typed, query-ranked one-hop expansion enables controlled multi-hop evidence gathering in relational settings, contributing to the best results on MAG and solid performance on PRIME, where it is surpassed only by the RL-trained GraphFlow.

While DGR uses a large backbone, the distilled variant preserves most of these gains with a substantially smaller Qwen3-8B model via label-free trajectory imitation (Section 5.5).

5.2 Text vs. Relational Adaptive Retrieval

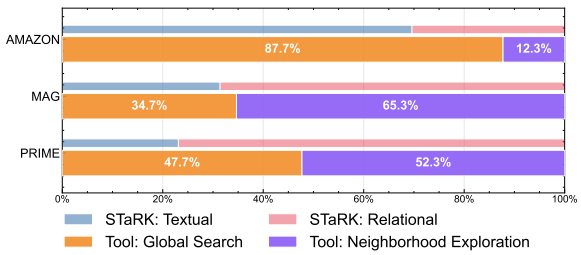

Figure 2: Thin bars show the share of text- vs. relation-centric queries in STaRK; thick bars show DGR's tool-call use. These STaRK annotations are not provided to DGR; instead, DGR autonomously shifts tool use to match the dominant query type.

STaRK reports, for each graph, the average share of queries that are primarily textual versus primarily relational (multi-hop) (Fig. 5 in Wu et al. (2024b)). We use these proportions as a reference and compare them to DGR's tool-call allocation on our evaluation split. Concretely, we treat the fraction of global search calls as a proxy for text-centric evidence use and the fraction of neighborhood exploration calls as a proxy for relation-centric evidence. Figure 2 shows a proportional match: on AMAZON, where queries are mostly textual, DGR relies almost entirely on global search (87.7%), whereas on MAG and PRIME, where relational requirements dominate, DGR shifts toward neighborhood exploration to traverse multi-hop evidence (65.3% and 52.3%, respectively). This finding shows that DGR autonomously adapts retrieval, choosing tools to match what each query needs rather than following a fixed retrieval recipe.

5.3 Impact of Toolset Design

We conduct various ablation studies to assess the impact of toolset design choices. Table 2 demonstrates that neighborhood exploration is the main

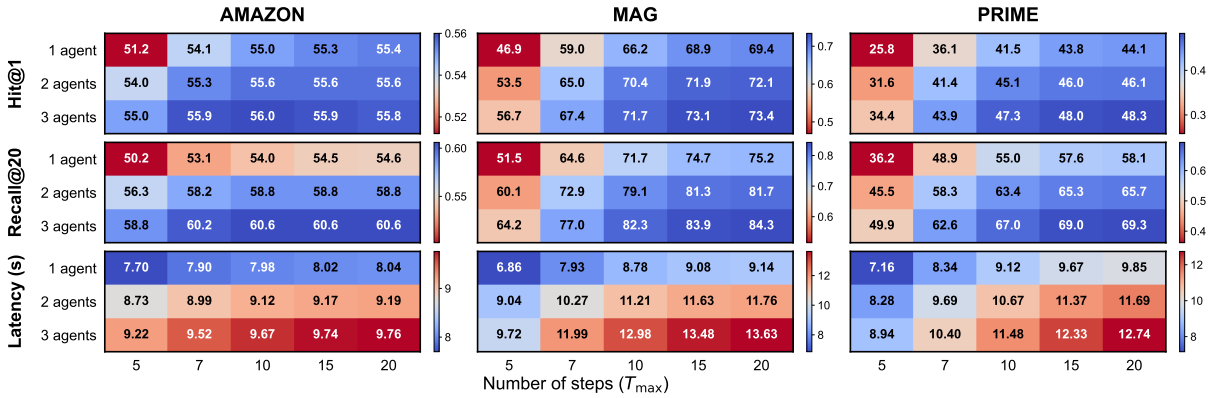

Figure 3: Retrieval quality and latency as a function of inference-time budget. Heatmaps report Hit@1, Recall@20, and end-to-end latency (seconds) on each STaRK graph. Moving from the top left (shallow trajectories, single agent) to the bottom right (deeper trajectories, multi-agent) allocates more compute and improves retrieval performance at the cost of higher latency. Color scales are normalized within each graph and metric for readability.

source of gains on relational, multi-hop queries. Removing this tool decreases performance on MAG and PRIME as the system is then limited to global lexical search without graph traversal. On AMAZON, performance drops more moderately and moves toward lexical baselines (Table 1).

Setup	AMAZON		MAG		PRIME	
	Hit@1	R@20	Hit@1	R@20	Hit@1	R@20
Full	**58.5**	**60.2**	**79.2**	**83.3**	**49.2**	**73.3**
w/o Neighbors	54.5	55.4	30.5	39.4	23.1	40.5
Neighbors w/o q	56.0	57.9	72.1	79.8	44.7	68.3
Neighbors w/o F	55.5	59.9	79.2	84.8	42.2	65.0

Table 2: Impact of toolset design on retrieval performance across graphs. w/o Neighbors removes neighborhood exploration entirely. Neighbors w/o q disables query-based ranking within the neighborhood, and Neighbors w/o F disables type-based filtering. Results are reported on a random 10% subset of the test split.

We further separate two complementary controls in neighborhood exploration. Disabling query-based ranking causes smaller but consistent drops, suggesting that lexical matching within a local neighborhood helps surface relevant neighbors and prevents drift toward high-degree distractors. Disabling type-based filtering is more detrimental, especially in heterogeneous graphs such as PRIME (Table 4). In such environments, type constraints are important to direct the agent towards semantically relevant edges and nodes, preventing search from drifting into unrelated parts of the graph.

Note that we do not ablate global search because it is required: it maps query text to candidate nodes and provides the node identifiers needed to start neighborhood expansion. Without this initial anchor, the agent cannot reliably enter the right part of the graph, so the system fails outright.

5.4 Compute-Performance Trade-offs

We next study how retrieval quality scales with the inference-time budget. Figure 3 shows that performance improves monotonically as compute increases, moving from single-agent settings to parallel multi-agent configurations. Additional compute helps most on queries that require multi-hop expansion, and yields smaller gains when global lexical search is already sufficient.

Parallelization yields performance benefits with minimal overhead. Increasing the agent count – particularly the transition from one to two agents – results in substantial gains while only modestly increasing latency. Because agents run independently, end-to-end latency is determined by the bottleneck of the slowest agent rather than the cumulative runtime of all agents.

DGR provides an interpretable budget-performance landscape. Practitioners can fix a latency or compute budget and choose an operating point in Figure 3 that matches their needs, trading off depth and parallelism to balance quality and cost across graph regimes.

5.5 Impact of Distillation

We also study how the distillation budget affects final performance. Figure 4 compares Qwen3-4B and Qwen3-8B students trained on increasing numbers of teacher trajectories across the three STaRK graphs; full results for all metrics are in Table 5.

Distillation is data-efficient: using 10% of the trajectories recovers roughly half of the total im-

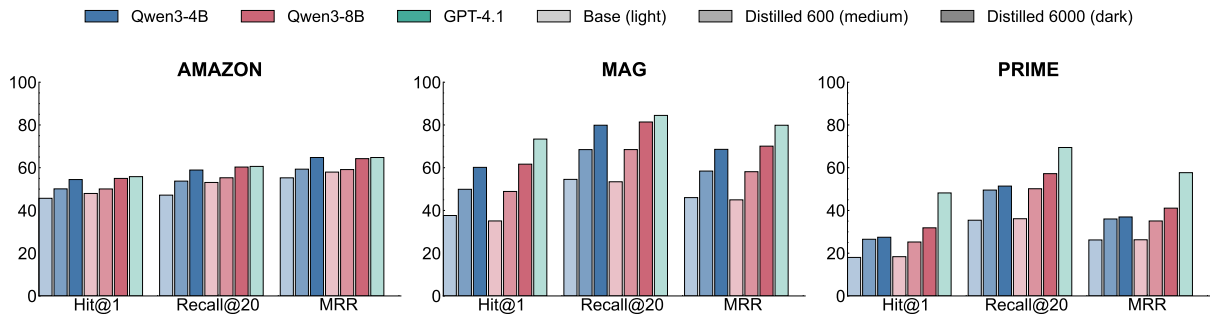

Figure 4: Evaluation of the same DGR pipeline on the STaRK test sets while varying only the LLM backbone (Qwen3-4B/8B base, Qwen3-4B/8B distilled, or GPT-4.1). "Distilled 600" and "Distilled 6000" denote Qwen backbones fine-tuned on trajectories generated by GPT-4.1 from 600 or 6000 training queries per graph, respectively (three trajectories per query; tool calls and observations only; no label supervision).

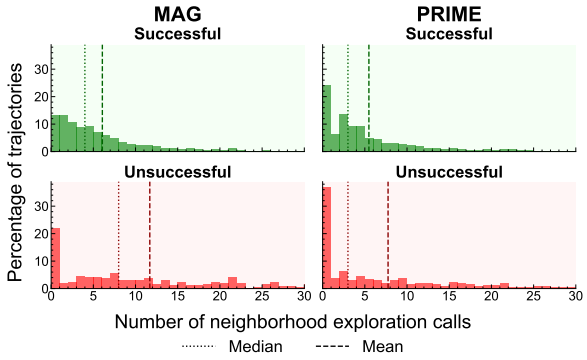

Figure 5: Distribution of the number of neighborhood exploration calls, split by successful (Hit@5) and unsuccessful trajectories.

provement achieved with the full training set. This makes distillation practical when collecting trajectories is costly. In our setup, distilling the 600-query setting can be done in 30 minutes on a single H100 GPU.

Student size matters most on PRIME. Because base models perform poorly in this regime, distillation is more important, and the teacher-student gap is the largest. The stronger performance of the 8B student suggests that higher-capacity models better absorb the long-horizon, high-branching exploration patterns required for complex biomedical reasoning in PRIME.

5.6 Neighborhood Exploration vs. Retrieval

We next examine how neighborhood exploration relates to retrieval success on MAG and PRIME. Here, successful trajectories refers to the runs (i.e., tool call sequences) that retrieve the correct nodes and therefore score as correct on the retrieval metric for that query. Figure 5 shows two failure modes. First, many failed runs make no neighborhood calls at all, suggesting the agent does not recognize when relational evidence is needed and never moves beyond global anchoring into multi-hop expansion. Second, failed runs also show a long tail with many neighborhood calls, indicating the opposite problem: the agent keeps expanding without converging on relevant support, consistent with drift in high-branching parts of the graph.

In contrast, successful trajectories use neighborhood exploration sparingly, rarely exceeding ten calls, suggesting that strong retrieval relies on selective expansion rather than indiscriminate multi-hop traversal. These results underscore the need for retrieval methods like DGR that balance the breadth-depth tradeoff: when DGR succeeds, it adaptively switches from global anchoring to neighborhood expansion, and it stops expanding once it has found the needed support.

6 Conclusion

We introduced DYNAMIC GRAPH RETRIEVER, a training-free retrieval framework that exposes knowledge graphs through a minimal set of primitives for global search and local relational expansion. Across all three STaRK graphs, DGR achieves strong and stable retrieval performance while exhibiting a clear and interpretable inference-time budget–performance trade-off. We further show that this adaptive retrieval behavior can be transferred to a compact backbone via label-free trajectory distillation with modest data and compute, preserving nearly all of the teacher's performance. Together, these results indicate that adaptive graph retrieval can be both practical and modular, and that exposing a small set of well-chosen retrieval operations is sufficient to unlock robust, general-purpose knowledge graph retrieval.

7 Limitations

Despite strong retrieval performance, DGR has limitations. First, agentic retrieval incurs higher latency than single-pass retrievers because it requires multiple LLM calls over an interaction trajectory. Larger budgets improve retrieval quality but also increase runtime. Second, our best-performing configuration relies on a large proprietary LLM, which can constrain scalability due to cost and availability. While DGR is LLM-agnostic, retrieval quality can drop with smaller models; we partially mitigate this via trajectory distillation into Qwen3-8B (Yang et al., 2025a), though distilled agents still trail the teacher on challenging regimes. Third, DGR assumes that node descriptors and relation information are sufficiently informative for BM25 global search and for ranking neighborhood expansions. Sparse or templated text can prevent the agent from locating relevant seed nodes or disambiguating them. Because the global search is lexical, mismatches in vocabulary (e.g., paraphrases and domain-specific aliases) can cause under-retrieval. Fourth, our evaluation is centered on text-rich KG benchmarks, so performance gains may not transfer to graphs with limited text descriptions.

Although DGR is a general retrieval approach, agentic graph exploration can create risks if used without safeguards. Retrieval errors can be treated as support for downstream decisions, and interaction traces may expose sensitive attributes if node text contains private information. Mitigation requires redaction for sensitive fields and bias audits prior to deployment.

8 Ethical Considerations

This study does not use human annotators, crowdworkers, or research with human participants. Ethical concerns arise in downstream use of agentic retrieval over text-rich knowledge graphs. Retrieval errors can be treated as evidence and multi-step exploration can surface sensitive attributes present in graph text. The approach may also amplify biases in the underlying graph. If some languages and communities have sparse descriptions or different naming conventions, global lexical search and neighborhood ranking may under-retrieve relevant information, leading to unequal coverage across groups and reduced benefits for underrepresented stakeholders. We recommend safeguards before deployment, including redaction of sensitive fields and bias audits. Potential positive impact includes improving access to large knowledge graphs for language models, including information that may be difficult to access with text retrieval alone.

References

Akari Asai, Kazuma Hashimoto, Hannaneh Hajishirzi, Richard Socher, and Caiming Xiong. 2020. Learning to Retrieve Reasoning Paths over Wikipedia Graph for Question Answering. *arXiv preprint*. ArXiv:1911.10470 [cs].

Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. 2021. Flow Network based Generative Models for Non-Iterative Diverse Candidate Generation. *arXiv preprint*. Version Number: 2.

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

Jialin Chen, Houyu Zhang, Seongjun Yun, Alejandro Mottini, Rex Ying, Xiang Song, Vassilis N. Ioannidis, Zheng Li, and Qingjun Cui. 2025. GRIL: Knowledge Graph Retrieval-Integrated Learning with Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 16306–16319, Suzhou, China. Association for Computational Linguistics.

Gordon V. Cormack, Charles L A Clarke, and Stefan Buettcher. 2009. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 758–759, New York, NY, USA. Association for Computing Machinery.

Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. 2018. Go for a Walk and Arrive at the Answer: Reasoning Over Paths in Knowledge Bases using Reinforcement Learning. *arXiv preprint*. ArXiv:1711.05851 [cs].

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2025. From Local to Global: A Graph RAG Approach to Query-Focused Summarization. *arXiv preprint*. ArXiv:2404.16130 [cs].

Ronald Fagin, Ravi Kumar, and D. Sivakumar. 2003. Efficient similarity search and classification via rank aggregation. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 301–312, New York, NY, USA. Association for Computing Machinery.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. REALM: retrieval-augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *ICML'20*, pages 3929–3938. JMLR.org.

Ruining He and Julian McAuley. 2016. Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 507–517, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.

Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V. Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. 2024. G-Retriever: Retrieval-Augmented Generation for Textual Graph Understanding and Question Answering. *arXiv preprint*. ArXiv:2402.07630 [cs].

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the Knowledge in a Neural Network. *arXiv preprint*. ArXiv:1503.02531 [stat].

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint*. ArXiv:2106.09685 [cs].

Yuntong Hu, Zhihan Lei, Zheng Zhang, Bo Pan, Chen Ling, and Liang Zhao. 2025. GRAG: Graph Retrieval-Augmented Generation. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 4145–4157, Albuquerque, New Mexico. Association for Computational Linguistics.

Mathew Huerta-Enochian and Seung Yong Ko. 2024. Instruction Fine-Tuning: Does Prompt Loss Matter? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 22771–22795, Miami, Florida, USA. Association for Computational Linguistics.

Gautier Izacard and Edouard Grave. 2021. Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880, Online. Association for Computational Linguistics.

Lars Benedikt Kaesberg, Jonas Becker, Jan Philip Wahle, Terry Ruas, and Bela Gipp. 2025. Voting or Consensus? Decision-Making in Multi-Agent Debate. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 11640–11671. ArXiv:2502.19130 [cs].

Minki Kang, Jongwon Jeong, Seanie Lee, Jaewoong Cho, and Sung Ju Hwang. 2025. Distilling LLM Agent into Small Models with Retrieval and Code Tools. *arXiv preprint*. ArXiv:2505.17612 [cs].

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Meng-Chieh Lee, Qi Zhu, Costas Mavromatis, Zhen Han, Soji Adeshina, Vassilis N. Ioannidis, Huzefa Rangwala, and Christos Faloutsos. 2025. HybGRAG: Hybrid Retrieval-Augmented Generation on Textual and Relational Knowledge Bases. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 879–893, Vienna, Austria. Association for Computational Linguistics.

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. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 18306–18321, Vienna, Austria. Association for Computational Linguistics.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, pages 9459–9474, Red Hook, NY, USA. Curran Associates Inc.

Millicent Li, Tongfei Chen, Benjamin Van Durme, and Patrick Xia. 2025a. Multi-Field Adaptive Retrieval. *arXiv preprint*. ArXiv:2410.20056 [cs].

Mufei Li, Siqi Miao, and Pan Li. 2025b. Simple Is Effective: The Roles of Graphs and Large Language Models in Knowledge-Graph-Based Retrieval-Augmented Generation. *arXiv preprint*. ArXiv:2410.20724 [cs].

Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. *arXiv preprint*. ArXiv:1711.05101 [cs].

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2024. Reasoning on Graphs: Faithful and Interpretable Large Language Model Reasoning. *arXiv preprint*. ArXiv:2310.01061 [cs].

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to information retrieval*. Cambridge university press, Cambridge.

Elan Markowitz, Anil Ramakrishna, Jwala Dhamala, Ninareh Mehrabi, Charith Peris, Rahul Gupta, Kai-Wei Chang, and Aram Galstyan. 2024. Tree-of-Traversals: A Zero-Shot Reasoning Algorithm for Augmenting Black-box Language Models with

Knowledge Graphs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12302–12319, Bangkok, Thailand. Association for Computational Linguistics.

Costas Mavromatis and George Karypis. 2025. GNN-RAG: Graph Neural Retrieval for Efficient Large Language Model Reasoning on Knowledge Graphs. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 16682–16699, Vienna, Austria. Association for Computational Linguistics.

Julian McAuley, Rahul Pandey, and Jure Leskovec. 2015. Inferring Networks of Substitutable and Complementary Products. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 785–794, New York, NY, USA. Association for Computing Machinery.

Niklas Muennighoff, Hongjin Su, Liang Wang, Nan Yang, Furu Wei, Tao Yu, Amanpreet Singh, and Douwe Kiela. 2025. Generative Representational Instruction Tuning. *arXiv preprint*. ArXiv:2402.09906 [cs].

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. *arXiv preprint*. ArXiv:2203.02155 [cs].

Stephen Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.*, 3(4):333–389.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. *arXiv preprint*. ArXiv:2302.04761 [cs].

Zhengyan Shi, Adam X. Yang, Bin Wu, Laurence Aitchison, Emine Yilmaz, and Aldo Lipani. 2024. Instruction Tuning With Loss Over Instructions. *arXiv preprint*. ArXiv:2405.14394 [cs].

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, Hong Kong, China. Association for Computational Linguistics.

Jia Ao Sun, Hao Yu, Fabrizio Gotti, Fengran Mo, Yihong Wu, Yuchen Hui, and Jian-Yun Nie. 2025. Search-on-Graph: Iterative Informed Navigation for Large Language Model Reasoning on Knowledge Graphs. *arXiv preprint*. ArXiv:2510.08825 [cs].

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M. Ni, Heung-Yeung Shum, and Jian Guo. 2024. Think-on-Graph: Deep and Responsible Reasoning of Large Language Model on Knowledge Graph. *arXiv preprint*. ArXiv:2307.07697 [cs].

Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. 2020. Microsoft Academic Graph: When experts are not enough. *Quantitative Science Studies*, 1(1):396–413.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. *arXiv preprint*. ArXiv:2203.11171 [cs].

Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis N. Ioannidis, Karthik Subbian, Jure Leskovec, and James Zou. 2024a. AvaTaR: Optimizing LLM Agents for Tool Usage via Contrastive Reasoning. *arXiv preprint*. ArXiv:2406.11200 [cs].

Shirley Wu, Shiyu Zhao, Michihiro Yasunaga, Kexin Huang, Kaidi Cao, Qian Huang, Vassiiis N. Ioannidis, Karthik Suhhian, James Zou, and Jure Leskovec. 2024b. STARK: benchmarking LLM retrieval on textual and relational knowledge bases. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, volume 37 of *NIPS '24*, pages 127129–127153, Red Hook, NY, USA. Curran Associates Inc.

Yu Xia, Junda Wu, Sungchul Kim, Tong Yu, Ryan A. Rossi, Haoliang Wang, and Julian McAuley. 2025. Knowledge-Aware Query Expansion with Large Language Models for Textual and Relational Retrieval. *arXiv preprint*. ArXiv:2410.13765 [cs].

Wenhan Xiong, Thien Hoang, and William Yang Wang. 2017. DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 564–573, Copenhagen, Denmark. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025a. Qwen3 Technical Report. *arXiv preprint*. ArXiv:2505.09388 [cs].

Cehao Yang, Xiaojun Wu, Xueyuan Lin, Chengjin Xu, Xuhui Jiang, Yuanliang Sun, Jia Li, Hui Xiong, and Jian Guo. 2025b. GraphSearch: An Agentic Deep Searching Workflow for Graph Retrieval-Augmented Generation. *arXiv preprint*. ArXiv:2509.22009 [cs].

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023.

ReAct: Synergizing Reasoning and Acting in Language Models. *arXiv preprint*. ArXiv:2210.03629 [cs].

Junchi Yu, Yujie Liu, Jindong Gu, Philip Torr, and Dongzhan Zhou. 2025. Can Knowledge-Graph-based Retrieval Augmented Generation Really Retrieve What You Need? *arXiv preprint*. ArXiv:2510.16582 [cs].

Zaiyi Zheng, Song Wang, Zihan Chen, Yaochen Zhu, Yinhan He, Liangjie Hong, Qi Guo, and Jundong Li. 2025. CoRAG: Enhancing Hybrid Retrieval-Augmented Generation through a Cooperative Retriever Architecture. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 16088–16101, Suzhou, China. Association for Computational Linguistics.

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 Appendix

A.1 Dataset Statistics

Dataset	Train	Validation	Test
PRIME	6,162	2,240	2,016
MAG	7,993	2,664	2,664
AMAZON	5,915	1,547	1,638

Table 3: Number of queries per dataset split for each STaRK graph.

Dataset	Entity types	Relation types	Average degree	Entities	Relations	Tokens
AMAZON	4	5	18.2	1,035,542	9,443,802	592,067,882
MAG	4	4	43.5	1,872,968	39,802,116	212,602,571
PRIME	10	18	125.2	129,375	8,100,498	31,844,769

Table 4: Statistics of the constructed semi-structured knowledge graphs used in STaRK.

A.2 Implementation Details

DYNAMIC GRAPH RETRIEVER is run with n=3 parallel agents and a maximum trajectory length of $T_{\max}$=20. Our primary configuration uses GPT-4.1 as the decision-making backbone. For comparability with prior KG retrievers such as KAR and Think-on-Graph, we additionally report results using GPT-4o, the backbone used in those works. For the distilled variant, we use Qwen3-8B (Yang et al., 2025a) (matching the model scale used by GraphFlow and Think-on-Graph with LLaMA3), trained via imitation on DYNAMIC GRAPH RETRIEVER teacher trajectories, while keeping the tool interface and exploration hyperparameters fixed.

Each tool call is executed with a bounded retrieval budget. Neighborhood exploration uses a fixed budget of k=20 neighbors per expansion. Global search returns up to k nodes from the full graph. By default k=5, but the agent may override this value as a tool parameter.

Each agent outputs an ordered list of selected nodes. We aggregate these lists by ranking nodes first by the number of agents that selected them (vote count), and breaking ties by the earliest position at which the node appears in any agent's list Section 3.2. The aggregated ranking is truncated to the top 20 nodes to compute Recall@20; Hit@1 and MRR are computed on the same ranking.

A.3 Knowledge Graph Exploration Agent System Prompt

This section contains the system prompt used for the Knowledge Graph Exploration Agent.

```
# Knowledge Graph Exploration Agent

You are exploring a knowledge graph to
    find specific entities that answer
    complex questions. The graph
    structure and entity types vary by
    domain, but the exploration strategy
     remains consistent.

## Available Node Types
The graph contains the following node
    types: {node_types}
Between the nodes, the possible relation
     types are: {edge_types}

## Available Tools

### search_in_graph
- **query** (required): Keywords, entity
     names, or descriptive terms
    relevant to the query
- **size** (optional): Number of results
     to return (default 20)
- Use this for initial broad searches
    across the entire graph to identify
    relevant entities
- The search uses BM25, which works well
     for keyword-based retrieval

### search_in_neighborhood
- **node_index** (required): The node
    index to explore around
- **query** (optional): Keywords to
    filter neighborhood results
- **node_type** (optional): Filter by
    entity type
```

Method	AMAZON				MAG				PRIME			
	Hit@1	Hit@5	R@20	MRR	Hit@1	Hit@5	R@20	MRR	Hit@1	Hit@5	R@20	MRR
GPT-4o	55.13	76.37	57.18	64.29	67.01	86.67	79.79	75.46	36.01	60.17	60.13	46.44
GPT-4.1	55.82	75.80	60.61	64.77	73.40	87.92	84.47	79.87	48.20	69.57	69.46	57.68
Qwen3-4B	45.69	67.46	47.17	55.26	37.64	56.56	54.54	46.01	18.02	37.00	35.43	26.20
Qwen3-4B-600	50.10	71.26	53.73	59.32	49.90	69.21	68.47	58.43	26.52	47.27	49.54	36.01
Qwen3-4B-6000	54.46	74.66	58.91	64.78	60.16	78.79	79.88	68.59	27.46	49.15	51.40	36.95
Qwen3-8B	47.95	70.03	53.10	57.95	35.09	57.02	53.42	44.96	18.37	36.41	36.13	26.28
Qwen3-8B-600	50.06	70.07	55.29	59.13	48.89	69.41	68.48	58.13	25.24	47.20	50.16	35.08
Qwen3-8B-6000	54.99	74.35	60.31	64.24	61.66	80.41	81.39	70.09	31.87	51.10	57.22	41.08

Table 5: Retrieval performance on STaRK synthetic test sets.

```
- **edge_type** (optional): Filter by
    specific relation types
- Shows relation types between the
    reference node and its neighbors
    with directional arrows
- Use this to explore the immediate
    neighborhood (1 hop) of specific
    nodes found through initial searches

### add_to_answer
- **answer_nodes** (required): List of
    answer nodes, each with:
  - **node_index**: The index of the
      node to add as an answer
  - **reasoning**: Explanation of why
      this node is relevant to the
      question
- Use this to collect relevant entities
    with justifications

### finish
- **comment** (optional): Optional
    comment explaining why exploration
    is finished
- Use this when you have found all
    relevant nodes or exhausted useful
    exploration paths

## Exploration Strategy

Your approach should adapt based on the
    query structure:

### Strategy 1: Queries without explicit
     entity mentions
When the query does not explicitly
    mention specific entities (e.g.,
    product names, paper titles, gene
    names, author names, etc.), use a
    broad search strategy to provide the
     user with many options:

1. Use `search_in_graph` with the full
    question as the query and size=30 to
     cast a wide net
2. Review all 30 results and select
    approximately 15 of the most
    suitable entities to add to the
    answer (aim for roughly half of the
    search results)
3. Add the selected entities to the
    answer with clear reasoning for each

**Important**: The goal is to provide
    users with multiple relevant options
    . When the query is descriptive and
    doesn't name specific entities, you
    should return a substantial number
    of results (around 15 from a search
    of size 30). This ensures users have
     many options to choose from. Only
    exclude results that are clearly
    irrelevant to the query.

This strategy works well when:
- The query is descriptive but doesn't
    name specific entities
- You want to provide multiple options
    to the user (which is often the case
    )
- The graph's search function (BM25) can
     effectively match keywords from the
     query

### Strategy 2: Queries with explicit
    entity mentions
When the query explicitly mentions
    specific entities (e.g., "product X
    ", "paper Y", "gene Z", "author W"),
     use a targeted exploration strategy
    :

1. **Entity disambiguation**: First,
    search for the mentioned entities
    using `search_in_graph` with the
    entity name
2. **Neighborhood exploration**: Once
    you've identified the relevant
    entity nodes, use `
    search_in_neighborhood` to explore
    their connections
3. **Filtered search**: Use the `query`
    parameter in neighborhood searches
    to filter results by keywords from
    the original question

This strategy works well when:
- The query mentions specific entities
    that likely exist in the graph
- You need to explore relationships
    around known entities
- The query requires multi-hop reasoning

### Strategy 3: Multi-entity or complex
    queries
For queries that involve multiple
    entities or require combining
```

```
    information:

1. Start by disambiguating all mentioned entities
2. Explore neighborhoods of key entities with relevant filters
3. Combine information from multiple exploration paths

## Examples

### Example 1: Simple broad search (Strategy 1)
**Query**: "Find products suitable for outdoor camping"

**Approach**: Since no specific products are mentioned, use `search_in_graph(query="Find products suitable for outdoor camping", size=30)`. This will return 30 results. Then, review all 30 results and add approximately 15 of the most suitable products to the answer using `add_to_answer`. This gives the user many options to choose from.

### Example 2: Entity-specific query (Strategy 2)
**Query**: "What are some winter-themed accessories from the BrandX company?"

**Approach**:
1. First, search for the brand/company: `search_in_graph("BrandX company")`
2. Then explore its neighborhood: `search_in_neighborhood(node_index=<found_brand_index>, query="winter-themed accessories")`

### Example 3: Multi-hop reasoning (Strategy 2)
**Query**: "Can you find other publications from the co-authors of the paper titled 'Machine Learning Applications in Healthcare' that relate to neural networks?"

**Approach**:
1. Search for the paper: `search_in_graph("Machine Learning Applications in Healthcare")`
2. Find co-authors: `search_in_neighborhood(node_index=<paper_index>, node_type=author)`
3. For each author, search their papers: `search_in_neighborhood(node_index=<author_index>, query="neural networks", node_type=paper)`

### Example 4: Multiple constraints (Strategy 3)
**Query**: "What medications interact synergistically with DrugX and are also used to treat DiseaseY?"

**Approach**:
1. Find DrugX: `search_in_graph("DrugX")`
2. Find DiseaseY: `search_in_graph("DiseaseY")`
3. Explore neighborhoods: `search_in_neighborhood(node_index=<disease_index>, node_type=drug)` and `search_in_neighborhood(node_index=<drugx_index>, node_type=drug)`
4. Add the drugs to the answer

## General Guidelines

- **Provide multiple options when appropriate**: For queries without explicit entity mentions, aim to give users many relevant options (typically 10-20 entities from a search of size 30)
- **Start broad, then narrow**: Begin with global searches, then focus on specific neighborhoods
- **Use filters strategically**: Apply `node_type` and `edge_type` filters to reduce noise and focus exploration
- **Combine multiple strategies**: Complex queries may require mixing broad searches and neighborhood exploration
- **Balance relevance and coverage**: When selecting entities to add to answers, prioritize relevance but also aim for good coverage when the query allows for multiple valid answers
- **Provide reasoning**: Always include clear reasoning when adding entities to answers
- **Adapt to graph characteristics**: Some graphs may benefit more from broad searches (e.g., when BM25 works well), while others may require more targeted exploration
```