DAG-RAG: MULTI-PATH HIERARCHICAL ENTITY GROUNDING FOR GRAPHRAG

Anonymous authors
Paper under double-blind review

ABSTRACT

Retrieval-augmented generation (RAG) enhances large language models by retrieving external knowledge, and GraphRAG further enhances this paradigm by organizing knowledge as graphs for structured reasoning. However, existing GraphRAG methods still struggle with grounding complex queries to relevant entities or subgraphs within a large graph space, leading to inefficient retrieval and unstable downstream generation. To address these issues, we propose DAG-RAG, a hierarchical GraphRAG framework that constructs a multi-granularity directed acyclic graph from the original knowledge graph and performs coarse-to-fine grounding with a path-conditioned router. The structured design also enables incremental maintenance by updating only affected clusters without full graph reconstruction or model retraining. Experiments show that DAG-RAG improves retrieval accuracy and downstream answer quality while reducing unnecessary graph traversal, demonstrating the effectiveness of hierarchical DAG grounding for scalable and adaptive GraphRAG.

1 INTRODUCTION

Retrieval-augmented generation (RAG) enhances language models by retrieving relevant external knowledge and incorporating it into the generation process. GraphRAG improves this paradigm by organizing external knowledge as a graph, where entities and relations provide structured context for more complex reasoning. But traditional GraphRAG approaches often suffer from poor retrieval performance on complex tasks due to a large search space and the challenging trade-off between accuracy and runtime. Thus, various methods have been developed to optimize subgraph retrieval for better trade-off between accuracy and runtime.

Research on subgraph retrieval methods in GraphRAG can be broadly classified into hierarchical retrieval (Hu et al., 2025; Guo et al., 2025; Cao et al., 2025; Hong et al., 2025; Wang et al., 2025; Zhu et al., 2025; Chen et al., 2026) and prior-based retrieval (Li et al., 2025b; Mavromatis & Karypis, 2025). However, due to inherent limitations of the original knowledge graph structure, focusing solely on subgraph retrieval optimization yields only limited improvements in overall information retrieval performance. Recent work has explored not only optimizing subgraph retrieval methods but also improving the underlying knowledge graph structure itself, including community-based graph (Edge et al., 2024; Wang et al., 2026; Yang et al., 2025), triple-linked evidence graphs (Wu et al., 2025) and HyperGraph structure (Luo et al., 2025a). Other specialized graph structures have been designed to address various challenges (Li et al., 2025a; Melo et al., 2025; Chen et al., 2026).

Nevertheless, GraphRAG still faces a key challenge: graph structures do not always support accurate entity grounding in a stable manner. Existing graph traversal methods are inherently entity-centric, requiring that target entities mentioned in a query be mapped to corresponding graph nodes or relevant communities before subsequent expansion and reasoning can occur (Liang et al., 2025; Yih et al., 2015; Edge et al., 2024; Mavromatis et al., 2025). Consequently, the ability to accurately map a query to the correct graph entities or subgraphs directly determines the scope of downstream retrieval and significantly influences the quality of the generated output. Achieving such precise alignment can be difficult because query expressions in real-world scenarios often do not naturally align with the node representations in the graph. This misalignment can arise from differences in granularity, for example, when a query uses a higher-level concept or a broader region that does not match a specific node, as well as from surface-form variations such as abbreviations, aliases, singu-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

lar–plural discrepancies or multilingual terms (Pons et al., 2024; Guellil et al., 2024). If the system relies predominantly on exact name matching or straightforward embedding similarity for entity alignment (Edge et al., 2024; Zheng et al., 2025; Lee et al., 2025; Huang et al., 2025), it becomes prone to mislocating entities or failing to locate them at all, leading to ambiguity propagation. These alignment failures can cause lower retrieval recall, break the relational chains needed for effective reasoning, and ultimately result in incomplete or incorrect generated answers.

In essence, the fundamental bottleneck of existing GraphRAG methods lies not only in the cost of graph construction or the complexity of reasoning, but also in their heavy dependence on stable entity grounding. While dynamic update strategies have been explored in conventional RAG settings (Luo et al., 2025b), most existing GraphRAG offline construction techniques lack effective support for incremental graph maintenance, particularly for deletion and revision (Zhang et al., 2025a).

To address these issues, we propose DAG-RAG, a hierarchical directed acyclic graph framework that organizes knowledge into hierarchy of clusters and performs coarse-to-fine grounding for efficient and accurate retrieval. Instead of relying on embedding similarity or directly feeding all graph entities into an LLM, DAG-RAG decomposes grounding into a sequence of local decisions over the DAG hierarchy. At each level, a lightweight router scores the children nodes, so that semantic judgment is preserved while the search space narrows progressively. We further condition each local decision on the branch through which a candidate is reached, adding a path-dependent residual to a level-local node score over a contrastively fine-tuned entity space, which lets the router separate locally competing branches and keep its decisions consistent across levels. Through this design, the proposed framework improves retrieval accuracy, removes the LLM judgments that entity grounding would otherwise require, and provides a more flexible structure for incremental graph maintenance.

The main contributions are summarized as follows:

1. We formulate entity grounding as a sequence of level-wise local decisions over a multi-parent DAG index, in which an entity is selected by the evidence accumulated over paths rather than by a single branch.
2. We instantiate each parent-child transition with a path-conditioned LM router that adds path residual and aggregate transitions with a posterior-weighted forward recursion.
3. We couple the traversal to a per-level candidate budget rather than a fixed beam width, so every query is scored under the same cost while the number of retained branches adapts to it, and grounding itself issues no LLM call, so the only online LLM tokens are those of answer generation.
4. On three QA benchmarks, DAG-RAG attains the competitive performance among five GraphRAG systems based on entity grounding and a vector-only RAG baseline, at a fraction of the token cost, and it stays robust as the indexed corpus grows.

2 PRELIMINARY

2.1 RAG AND GRAPHRAG

RAG follows a two-step *retrieve-then-generate* pipeline, where a retriever first selects query-relevant evidence from an external corpus and a generator then produces the answer conditioned on it. Let $\mathcal{Q}$ and $\mathcal{D}$ denote the query space and an external corpus, respectively. Given a query $q \in \mathcal{Q}$, the retriever evaluates each document $d_i \in \mathcal{D}$ using the relevance function $p_\eta(d_i \mid q)$, parameterized by η, and returns the top-k most relevant documents as retrieved evidence:

$$\mathcal{D}_r = \arg \max_{\substack{\mathcal{D}_r \subseteq \mathcal{D} \\ |\mathcal{D}_r| = k}} \sum_{d_i \in \mathcal{D}_r} p_\eta(d_i \mid q), \tag{1}$$

where $\mathcal{D}_r \subseteq \mathcal{D}$ with $|\mathcal{D}_r| = k$ is the set of retrieved documents most relevant to q. The retrieved set $\mathcal{D}_r$ is then serialized and passed to the generator to produce the final answer.

Rather than retrieval over isolated text units, GraphRAG can be abstracted as structure-aware retrieval over a text-attributed graph

$$\mathcal{G} \triangleq (\mathcal{V}, \mathcal{R}, \mathcal{X}), \tag{2}$$

where $\mathcal{V}$ is the entity set, $\mathcal{R}$ is the relation set, and $\mathcal{X}$ denotes textual content associated with the graph, such as source chunks, node and edge descriptions, or higher-level summaries.

At inference time, GraphRAG first grounds the query to a set of seed entities $\mathcal{V}_q$, scored by a grounding relevance $p_\psi(v_i \mid q, \mathcal{V})$ parameterized by ψ. Analogous to the retrieval objective in equation 1, grounding selects the top-k entities that maximize their total relevance to the query q:

$$\mathcal{V}_q = \arg \max_{\mathcal{V}_q \subseteq \mathcal{V}, |\mathcal{V}_q|=k} \sum_{v_i \in \mathcal{V}_q} p_\psi(v_i \mid q, \mathcal{V}), \tag{3}$$

where $\mathcal{V}_q \subseteq \mathcal{V}$ with $|\mathcal{V}_q| = k$ is the grounded seed entity set, analogous to $\mathcal{D}_r$ in equation 1. The grounding relevance p_ψ admits two implementations. *Flat dense retrieval* (Karpukhin et al., 2020; Izacard et al., 2022; Xiao et al., 2024) efficiently scores entities independently by query–entity similarity in a shared embedding space, but overlooks contextual semantics (Weller et al., 2026) and, as the graph grows, forces gold entities to compete with more semantically similar nodes (Reimers & Gurevych, 2021). Alternatively, an *LLM-based semantic judge* (Sun et al., 2023; Ding et al., 2024) scores entities using richer query and graph context (Sun et al., 2024), enabling judgments beyond surface-level similarity; however, directly providing the entire graph exceeds practical LLM context and computational budgets (Fatemi et al., 2024; Liu et al., 2024).

Starting from $\mathcal{V}_q$, GraphRAG expands the seed entities along $\mathcal{R}$ into a query-relevant subgraph $\mathcal{G}_r = (\mathcal{V}_r, \mathcal{R}_r, \mathcal{X}_r)$. It selects the subgraph that is jointly most relevant to the query among those anchored at the grounded seed entities:

$$\mathcal{G}_r = \arg \max_{\mathcal{G}_r \subseteq \mathcal{G}, \mathcal{V}_q \subseteq \mathcal{V}_r, |\mathcal{V}_r|=k} p_\theta(\mathcal{G}_r \mid q)\,, \tag{4}$$

where p_θ scores subgraph rather than independent units, and the constraint $\mathcal{V}_q \subseteq \mathcal{V}_r$ makes the grounded seeds the entry point of the expansion. The retrieved subgraph is finally serialized and passed to the generator as $\hat{a} = \arg\max_a p_\phi(a \mid q, \mathcal{G}_r)$. Rather than retrieving isolated documents, GraphRAG changes evidence to query-relevant subgraph, which contains structural and relational content for generation. However, since the expansion is anchored at $\mathcal{V}_q$, inaccurate grounding propagates to the retrieved subgraph, making entity grounding critical to GraphRAG.

3 Algorithm

3.1 Problem Formulation

To bridge the gap between flat dense grounding and full-graph LLM grounding, we group the entities into coarse-to-fine levels $\mathcal{V}^1, \mathcal{V}^2, \ldots, \mathcal{V}^{n-1}$ above the entity level $\mathcal{V}^n \triangleq \mathcal{V}$, which together form a directed acyclic graph (DAG) index $\mathcal{H} \triangleq (\mathcal{V}^1, \mathcal{V}^2, \ldots, \mathcal{V}^{n-1}, \mathcal{V}^n)$, where a node at level $\ell < n$ is a cluster of entities and a node at level n is an entity itself. Since a finer-level node may belong to multiple coarser-level nodes, the index forms a DAG rather than a tree, so the same node can be reached through more than one grounding path.

Given a DAG index $\mathcal{H}$, grounding is no longer a single decision over the whole entity set as in equation 3, but a sequence of level-wise decisions along the index. Let $v^{(\ell)} \subseteq \mathcal{V}^\ell$ denote the node set retrieved at level ℓ, with $v^{(n)} = \mathcal{V}_q$. Since the intermediate node sets $v^{(1)}, \ldots, v^{(n-1)}$ are latent, we marginalize over all of them, so that each candidate entity accumulates the probability mass of every level-wise path that reaches it:

$$\mathcal{V}_q = \arg \max_{\substack{\mathcal{V}_q \subseteq \mathcal{V}^n \\ |\mathcal{V}_q|=k}} \sum_{v_q \in \mathcal{V}_q} \sum_{v^{(1)}, \ldots, v^{(n-1)}} p_\psi\Big(v^{(1)} \mid q\Big)\, p_\psi\Big(v^{(2)} \mid v^{(1)}, q\Big) \cdots p_\psi\Big(v_q \mid v^{(n-1)}, q\Big)\,, \tag{5}$$

where each factor is a local decision that only has to discriminate among the children of the previously retrieved nodes. Writing a complete path i that reaches a selected entity $v_q \in \mathcal{V}_q$ as $\pi_i(v_q) = \Big(v_i^{(1)}, v_i^{(2)}, \ldots, v_i^{(n-1)}, v_q\Big)$, where each $v_i^{(\ell)} \in \mathcal{V}^\ell$ is a single node and every consecutive pair is a parent–child edge of $\mathcal{H}$, the grounding probability of v_q is obtained by summing the probability mass over the set $\Pi(v_q) = \{\pi_i(v_q)\}_i$ of all such paths. $\mathcal{H}$ is a DAG rather than a tree, making $|\Pi(v_q)| \geq 1$. So a node is not pruned by a single incorrect decision at a coarser level.

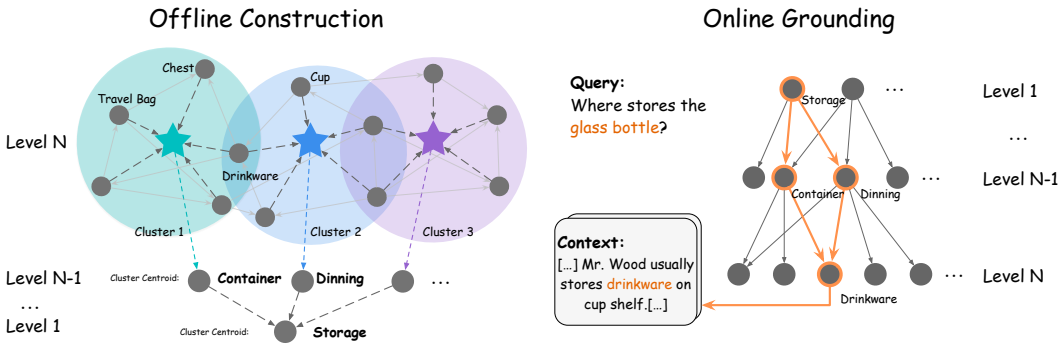

Figure 1: DAG-RAG offline construction and online grounding pipeline.

Exact marginalization over all grounding paths is intractable, since the number of DAG-consistent paths grows exponentially with the depth of $\mathcal{H}$. We therefore restrict the latent sums to the node sets that the traversal retains at each level, which are selected level by level at inference time. Once all the latent node sets are determined, an entity's grounding probability is approximated by the product of the local decision probabilities along the retained levels, and the seed entities are selected as

$$\mathcal{V}_q \approx \arg \max_{\substack{\mathcal{V}_q \subseteq \mathcal{V}^n \\ |\mathcal{V}_q|=k}} \sum_{v_q \in \mathcal{V}_q} p_\psi\Big(v^{(1)} \mid q\Big)\, p_\psi\Big(v^{(2)} \mid v^{(1)}, q\Big) \cdots p_\psi\Big(v_q \mid v^{(n-1)}, q\Big)\,, \tag{6}$$

where $v^{(1)}, \ldots, v^{(n-1)}$ are fixed to the retained node sets instead of being summed over as in equation 5. By omitting the probability mass of discarded paths, this approximation lower-bounds the full marginal grounding probability and recovers it exactly when every level is retained in full. Instead of judging every entity in the full graph, the grounding model only needs to make local semantic decisions conditioned on the previously retrieved coarser-level nodes, which preserves the semantic advantage of LLM-based grounding while avoiding the infeasible requirement of presenting the entire graph index to LLM at once.

3.2 Hierarchical Grounding

To solve the grounding objective of Sec. 3.1, we build offline a hierarchical index that organizes the entity set into coarse-to-fine levels, we then ground it online by a customized level-wise routing that traverses this index, forming the seed set $\mathcal{V}_q$ at the entity level, as shown in Fig. 1.

Offline construction. We use an LLM to extract entities and relations from the corpus and organize the resulting entity set into a hierarchical index. Each entity forms a leaf node holding the normalized entity embedding produced by a fixed fine-tuned entity encoder, and the hierarchy is built bottom-up by recursively applying spherical k-means to the representations of one level, with every node above the leaves represented by the normalized centroid of its children. Clustering in the same normalized space in which the router later compares a query with its candidates keeps the granularity of the index aligned with the geometry of the grounding decisions. To avoid the brittleness of assigning each node to a single parent, we retain multiple parent links whose similarity is within a threshold of the best-matching parent, forming a DAG rather than a tree. The redundancy introduced by these links is what makes an early mistake recoverable: a relevant entity stays reachable as long as any one of its ancestors survives.

A node stores its normalized representation together with its parent and child links, from which the set of nodes it covers is recovered by following the child links recursively. Unlike the community detection used in prior GraphRAG systems, this hierarchy is not meant to expose relational structure; it organizes the search space of grounding, and its levels are defined by semantic proximity rather than by connectivity. Because the index is obtained from encoder representations through matrix operations, it is inexpensive to build and can be rebuilt while remaining globally consistent.

Online grounding. Given a query q, grounding proceeds top-down through the hierarchy, examining at each level only the children of the nodes retained above it without ever scoring the full entity

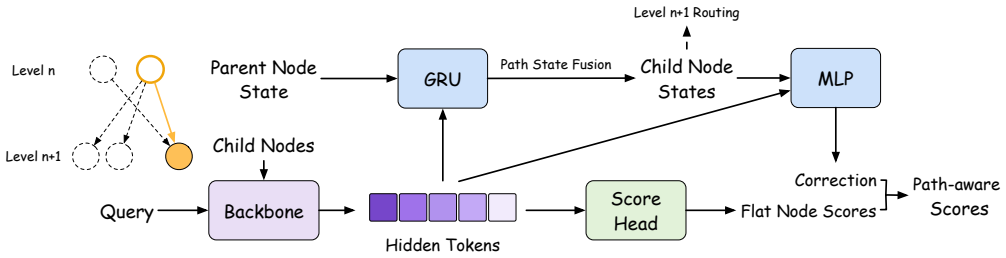

Figure 2: Path-conditioned routing at level n of the hierarchy.

set. Since different retained nodes may share children, we rank nodes by accumulated path evidence and retain them until the deduplicated candidate set reaches the budget for the next level, keeping computation bounded while adapting the number of explored branches. At the leaf level, the surviving entities are ranked by the accumulated evidence to form $\mathcal{V}_q$. Evidence arriving from several coarse-to-fine branches thus reinforces the same entity, which is exactly the behaviour a tree-shaped index cannot express.

Scoring a candidate by its query similarity alone would give it the same relevance under every parent, so we design a router that conditions each transition on the branch through which the candidate is reached, as shown in Fig. 2. To keep this traversal cheap, the router operates on the precomputed entity and the query representations, together with the structural features of the candidate nodes, instead of assembling a textual sequence for each candidate. Every candidate node receives its own contextualized hidden state, from which the score head reads the transition score, with cost scaling by candidate count rather than textual length. Neural scoring is executed under BF16 autocast, while the resulting logits and the accumulation of path evidence are maintained in FP32, which preserves the small score differences that separate competing branches.

3.3 Path-Conditioned LM Router

The traversal of Sec. 3.2 visits only a part of the DAG, so grounding aggregates probability mass only over retained paths. Writing $\widehat{\Pi}(v_q) \subseteq \Pi(v_q)$ for the paths that reach an entity v_q and are retained by the traversal, the set returned at the leaf level is

$$\mathcal{V}_q = \arg \max_{\mathcal{V}_q \subseteq \mathcal{V}^n, |\mathcal{V}_q|=k} \sum_{v_q \in \mathcal{V}_q} \sum_{\pi \in \widehat{\Pi}(v_q)} p_\psi(\pi \mid q)\,, \tag{7}$$

where entities that no retained path reaches contribute nothing. This budgeted objective selects entities by the total probability mass of their surviving branches rather than that of the single best branch. Evaluating it calls for two ingredients, which this section provides in turn: a transition probability for every retained parent–child edge, and an accumulation of these transitions that never enumerates paths.

Each grounding path is factorized into local decisions at successive parent–child transitions. We instantiate these decisions with the router in Fig. 2, which combines two complementary signals: a *node score* that measures the relevance of a candidate within the current level, and a *path correction* that accounts for the branch through which that candidate is reached. Each node uses a precomputed representation from the hierarchy's embedding space, with the entity encoder contrastively fine-tuned and then frozen. The backbone jointly encodes the query and the candidates of a level, producing a contextualized state $h_i^{(\ell)}$ for each candidate $v_i^{(\ell)} \in \mathcal{V}^\ell$, whose parents are the nodes of the retained set $v^{(\ell-1)}$. A scalar head reads the flat node score $s_i^{(\ell)}$ from $h_i^{(\ell)}$.

The router carries a fixed-dimensional state $z_j^{(\ell-1)}$ for every retained node $v_j^{(\ell-1)}$. We introduce a virtual root r above the coarsest level and initialize its state with learned parameters. For each retained edge, the parent state is updated with the contextualized child representation as $\tilde{z}_{ji}^{(\ell)} = \mathrm{GRU}\left(h_i^{(\ell)}, z_j^{(\ell-1)}\right)$. An edge network then turns this path state into a residual correction to the

node score. The transition probability from parent $v_j^{(\ell-1)}$ to child $v_i^{(\ell)}$ is then

$$p_\psi\Big(v_i^{(\ell)} \mid v_j^{(\ell-1)}, q\Big) = \frac{\exp\Big(s_i^{(\ell)} + D_\psi\big(\widetilde{z}_{ji}^{(\ell)}, h_i^{(\ell)}\big)\Big)}{\sum_{i'} \exp\Big(s_{i'}^{(\ell)} + D_\psi\big(\widetilde{z}_{ji'}^{(\ell)}, h_{i'}^{(\ell)}\big)\Big)}, \tag{8}$$

where i' ranges over the complete child set of $v_j^{(\ell-1)}$. We zero-initialize the output layer of D_ψ, so the router initially reduces to a softmax over the flat node scores and learns only the path-dependent residual needed to reorder siblings.

These edge-local transition probabilities must next be combined across DAG, since the same node may receive probability mass from several retained parents. Rather than keeping a separate hidden state for every complete path, we aggregate both the probability mass and the recurrent states at each node. We write $\hat{p}_\psi(\cdot \mid q)$ for the marginal of a node under the retained paths only, that is the total probability of the paths that reach it and survive the traversal. Starting from $\hat{p}_\psi(r \mid q) = 1$ at the virtual root, the forward recursion is

$$\begin{aligned}
\hat{p}_\psi\Big(v_i^{(\ell)} \mid q\Big) &= \sum_j \hat{p}_\psi\Big(v_j^{(\ell-1)} \mid q\Big)\, p_\psi\Big(v_i^{(\ell)} \mid v_j^{(\ell-1)}, q\Big)\,, \\
\hat{p}_\psi\Big(v_j^{(\ell-1)} \mid v_i^{(\ell)}, q\Big) &= \frac{\hat{p}_\psi\Big(v_j^{(\ell-1)} \mid q\Big)\, p_\psi\Big(v_i^{(\ell)} \mid v_j^{(\ell-1)}, q\Big)}{\hat{p}_\psi\Big(v_i^{(\ell)} \mid q\Big)}, \\
z_i^{(\ell)} &= \sum_j \hat{p}_\psi\Big(v_j^{(\ell-1)} \mid v_i^{(\ell)}, q\Big)\, \widetilde{z}_{ji}^{(\ell)}.
\end{aligned} \tag{9}$$

Incoming GRU states $\widetilde{z}_{ji}^{(\ell)}$ are merged using posterior weights proportional to their probability contributions. At the leaf level, this recursion accumulates the total retained path mass used for entity selection, evaluating the budgeted grounding objective by dynamic programming while keeping only one marginal and one path state per retained node. At intermediate levels, nodes are retained according to $\hat{p}_\psi(\cdot \mid q)$ under the candidate-budget rule of Sec. 3.2, and at the leaf level the k entities with the largest marginal form $\mathcal{V}_q$.

Finally, we train the router on the oracle seed set $\mathcal{V}_q^*$ of a query, the entities that reverse grounding links to one of its supporting sentences or to its answer. A node above the leaf level is positive if at least one entity of $\mathcal{V}_q^*$ lies among its descendants. Training uses fixed cosine-routed frontiers with positive-node recovery and complete child sets for each retained parent (§A.3).

Let $\mathcal{G}_j^{(\ell)}$ denote the positive children of a retained node $v_j^{(\ell-1)}$ (including the virtual root), the continuations that keep at least one entity of $\mathcal{V}_q^*$ reachable. Since several children may cause oracle entities, we maximize their total transition probability rather than forcing a single positive continuation:

$$\mathcal{L}_{\text{route}} = -\,\mathbb{E}_{j:\,|\mathcal{G}_j^{(\ell)}|>0}\left[\log \sum_{i \in \mathcal{G}_j^{(\ell)}} p_\psi\Big(v_i^{(\ell)} \mid v_j^{(\ell-1)}, q\Big)\right]. \tag{10}$$

To align training with the final entity ranking, we additionally supervise the aggregated leaf marginals,

$$\mathcal{L}_{\text{leaf}} = -\frac{1}{|\mathcal{V}_q^*|} \sum_{v \in \mathcal{V}_q^*} \log \frac{\hat{p}_\psi(v \mid q)}{\sum_{v'} \hat{p}_\psi(v' \mid q)}, \qquad \mathcal{L} = \mathcal{L}_{\text{route}} + \lambda \mathcal{L}_{\text{leaf}}, \tag{11}$$

where v' ranges over the entity candidates that reach the leaf level. The routing loss preserves valid continuations, while the leaf loss supervises entity selection across branches. We jointly optimize the backbone, score head, recurrent cell, and edge network, while keeping the entity encoder fixed.

4 EXPERIMENTAL RESULTS

4.1 EXPERIMENTAL SETUP

We evaluate DAG-RAG, a complete GraphRAG method with hierarchical DAG-based entity grounding, on three question answering benchmarks: **HotpotQA** (Yang et al., 2018), an open-domain benchmark built on Wikipedia; **MuSiQue** (Trivedi et al., 2022), which requires compositional reasoning over multiple supporting facts; and **PopQA** (Mallen et al., 2023), an open-domain, entity-centric benchmark for factual question answering about long-tail entities. These datasets provide complementary settings for testing entity grounding and evidence retrieval. For each dataset we build the index from the passages of a $1{,}000$-question pool and test the random 250 questions for evaluations; the encoder and the router are trained on a disjoint pool drawn from HotpotQA and MuSiQue only, so every PopQA result is a zero-shot transfer result.

To evaluate DAG-RAG as a complete system rather than an isolated module, the main comparison places DAG-RAG and the baseline RAG systems in the same end-to-end setting. Every reported number therefore reflects the full question-to-answer pipeline of each system.

The experiments are organized into four parts. First, we compare DAG-RAG end to end with a purely vector-based Vanilla RAG baseline, and five representative GraphRAG systems based on entity grounding: KAG (Liang et al., 2025), LightRAG (Guo et al., 2025), SiReRAG (Zhang et al., 2025b), Microsoft GraphRAG (Edge et al., 2024) and HippoRag(v1) (Gutiérrez et al., 2024). Second, we open up the routing score itself and measure, level by level, how much of each decision comes from the level-local node score and how much from the path correction. Third, we evaluate passage-level grounding for DAG-RAG and the baseline methods as the indexed corpus grows while keeping the question set fixed. Fourth, we conduct an ablation study that quantifies the contribution of the individual components of DAG-RAG and an entity-level grounding recall across a range of retrieval budgets.

4.2 MAIN EXPERIMENT

The main experiment compares DAG-RAG with representative systems under the end-to-end protocol described above. Each system runs its own complete pipeline, so the comparison reflects the joint behaviour of retrieval and generation rather than the quality of any single component in isolation. In DAG-RAG the router is the only new component: the entities are expanded by one hop and the corresponding passages are then reranked by vector similarity and graph connectivity to the query, and the top-ranked ones go to the same generator as baselines. Every downstream stage is conditioned on the router's output, making Table 1 reflecting comparison of grounding accuracy.

We report five metrics covering answer quality and inference cost: Exact Match (EM), token-level F1, LLM Judge Score (LJS), Tokens, and End-to-end Time(Time). EM and F1 measure the quality of the generated answers, while LJS uses an LLM to judge whether the generated answer contains the meaning of the reference answer and assigns a binary score of 0 or 1; Tokens and Time characterize the cost a system pays for the answer quality it achieves.

DAG-RAG delivers the competitive overall answer quality across the three datasets and at the second-lowest overall cost among the graph-based systems.

4.3 ROUTING SCORE ANALYSIS

The transition of equation 8 scores a candidate with two terms: the flat node score $s_i^{(\ell)}$, which sees only the query and the candidates competing at the current level, and the path correction $\Delta_i^{(\ell)}$ which sees the branch through which the candidate is reached.

We run the trained router over the evaluation questions and compare the flat scores and path corrections of gold and non-gold candidates at each routing level. Table 2 reports, per level, the mean gap μ and dispersion σ of the correction, the mean flat scores of gold and non-gold candidates before the correction is applied, and Kendall's τ between the correction and the flat score over the gold candidates, which measures how much of the correction's ranking is already contained in the flat score.

Table 1: End-to-end performance comparison.

Dataset	Method	EM	F1	LJS	Tokens	Time (s)
HotpotQA	Vanilla RAG	0.448	0.609	0.736	3.2k	6.9
	KAG	**0.572**	0.713	0.836	110.1k	85.3
	LightRAG	0.532	0.684	0.796	14.2k	36.2
	SiReRAG	0.328	0.441	0.516	4.1k	59.2
	MS GraphRAG	0.420	0.542	0.708	6.8k	35.5
	HippoRAG	0.452	0.580	0.672	**1.9k**	47.9
	DAG-RAG	0.544	**0.726**	**0.852**	3.1k	**25.5**
MuSiQue	Vanilla RAG	0.176	0.267	0.336	2.9k	10.8
	KAG	0.328	**0.490**	0.500	29.1k	114.0
	LightRAG	0.204	0.307	0.408	12.1k	42.1
	SiReRAG	0.108	0.124	0.152	3.1k	45.4
	MS GraphRAG	0.096	0.184	0.364	4.7k	32.4
	HippoRAG	0.284	0.407	0.468	**1.6k**	**10.5**
	DAG-RAG	**0.336**	0.436	**0.532**	3.0k	19.1
PopQA	Vanilla RAG	0.336	0.488	0.696	3.1k	7.1
	KAG	0.368	0.486	0.696	15.7k	22.5
	LightRAG	0.336	0.471	0.676	15.2k	25.4
	SiReRAG	0.340	0.484	0.680	3.8k	18.7
	MS GraphRAG	0.280	0.387	0.608	6.4k	24.4
	HippoRAG	0.356	0.474	0.668	**1.9k**	**2.9**
	DAG-RAG	**0.392**	**0.515**	**0.716**	3.3k	5.7

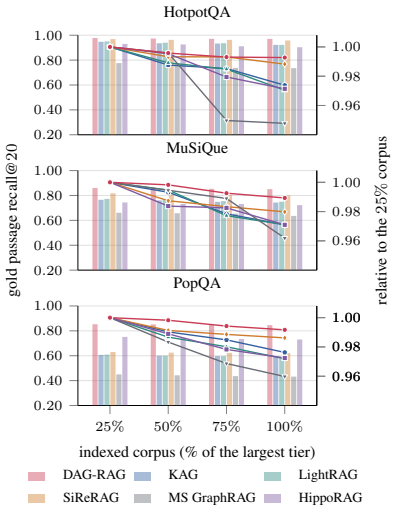

Figure 3: Passage-level recalling performance across corpus scales.

Across the evaluated datasets, corrections provide clear gains in routing performance, and gold candidates receive larger path corrections at deeper routing levels.

4.4 Corpus-Scale Experiment

To test robustness to corpus growth, we construct a sequence of nested corpora from the same source collection and evaluate the same fixed question set at every scale.

We measure grounding here at the passage level rather than the entity level, with budget of top-20, enabling a fair comparison across baselines. Figure 3 reports the raw fraction as bars on the left axis, and each system's value relative to smallest-corpus result as curves on the right axis.

DAG-RAG achieves the highest passage recall rates across the datasets and scales, also exhibits the smallest relative decline from the smallest to the largest corpus.

4.5 Entity Recall Analysis and Ablation Study

To isolate the contribution of DAG-RAG each pipeline stage, Table 3 removes or replaces one stage of the pipeline at a time and reads the effect end to end, in the same metrics as the main experiment4.2.

Table 2: Level-wise decomposition of the routing score into node scores and path corrections.

Dataset	Level	Correction Δ		Flat score s		$\tau(s, \Delta)$
		μ	σ	gold	non-gold	
HotpotQA	L0	2.303	1.322	4.507	3.410	0.618
	L1	3.315	1.629	4.656	3.279	0.252
	L2	4.804	1.147	4.869	3.025	0.045
MuSiQue	L0	1.620	0.543	4.532	3.450	0.510
	L1	2.079	0.886	4.573	3.361	0.277
	L2	3.628	1.325	4.558	3.153	-0.006
PopQA	L0	1.677	1.341	4.785	3.580	0.592
	L1	2.304	1.206	4.770	3.503	0.278
	L2	3.434	2.282	4.764	3.362	0.082

Pipeline ablation configurations. The "oracle gold entities" configuration replaces the grounding stage entirely with gold entities from the dataset's annotated passages and runs the remainder of DAG-RAG unchanged. It is not an general ablation but an upper bound for entity grounding method on what the downstream stages can do given perfect grounding. The "w/o 1-hop expansion" configuration removes the one-hop neighbourhood expansion applied to the routed entities before passage scoring, so only the entities the router returns can contribute evidence. The "w/o graph passage score" configuration drops the contribution the graph structure makes to the

Table 3: End-to-end ablation of DAG-RAG pipeline components.

Dataset	Configuration	EM	F1	LJS	Tokens	Retrieval (ms)
HotpotQA	full model	0.544	0.726	0.852	3.1k	532
	oracle gold entities	0.564	0.743	0.864	3.1k	3
	w/o 1-hop expansion	0.504	0.654	0.804	3.1k	528
	pure vector retrieval	0.500	0.659	0.772	3.1k	64
	flat router input	0.524	0.681	0.832	3.1k	4,061
	w/o graph passage score	0.540	0.708	0.824	3.2k	530
MuSiQue	full model	0.336	0.436	0.532	3.0k	700
	oracle gold entities	0.356	0.446	0.576	3.0k	21
	w/o 1-hop expansion	0.256	0.342	0.436	2.9k	701
	pure vector retrieval	0.252	0.351	0.448	3.0k	22
	flat router input	0.308	0.397	0.496	3.0k	4,440
	w/o graph passage score	0.316	0.406	0.536	3.0k	703
PopQA	full model	0.392	0.515	0.716	3.3k	749
	oracle gold entities	0.432	0.549	0.740	3.3k	6
	w/o 1-hop expansion	0.388	0.485	0.684	3.3k	747
	pure vector retrieval	0.352	0.461	0.644	3.3k	7
	flat router input	0.352	0.476	0.692	3.3k	4,383
	w/o graph passage score	0.372	0.510	0.704	3.2k	750

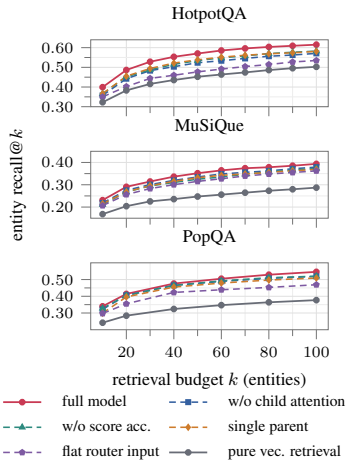

Figure 4: Entity recall ablation across retrieval budgets.

final passage ranking, scoring each passage purely by its own semantic similarity to the query. The "pure vector retrieval" configuration replaces routed entity search with cosine similarity against the whole entity index, keeping the top k entities. It shares the encoder and the entity index with the router, so the difference reflects the routing decision rather than any change in representation. The "flat router input" configuration keeps the router and its parameters but presents the entire node set to it in one step instead of descending the DAG level by level, so the scorer is unchanged and only the candidate narrowing is removed. The last two configurations are the same settings that appear in Figure 4 under the same names, evaluated here end to end rather than at the entity level.

To evaluate the router and its components, Figure 4 computes the gold entity recall rate across a range of retrieval budgets, and removes one router component at a time to compare their contributions. The entity-level results use the same evaluation questions and the same indices over three datasets. Every variant shares the encoder and the entity index with the full model, so the gap between any two of them reflects the ablated component alone.

Router ablation configurations. The "w/o child attention" variant removes the self-attention over the child roster during router training, so each candidate is scored on its own rather than against the other candidates available at the same level. This variant measures the contribution of letting sibling candidates compete jointly. The "w/o score accumulation" variant removes the recurrent accumulation of the correction score along the traversal path, so a node is scored only by the current level and carries no evidence from the levels above it. This variant measures the contribution of propagating routing evidence down the path. The "single parent" variant restricts each lower-level node to a single parent, replacing the multi-parent DAG structure with a tree while keeping the remaining procedure unchanged. This variant measures the benefit of allowing entities to be reached through multiple valid hierarchical paths.

The full model achieves higher exact-match accuracy and answer overlap than every pipeline ablation expect oracle gold entities setup. It also maintains the highest entity recall across the evaluated retrieval budgets compared with all router ablations and pure vector retrieval.

5 Conclusion

In this paper, we presented DAG-RAG, a hierarchical GraphRAG framework that combines a multi-parent DAG with path-conditioned, task-aware routing to ground complex queries from coarse clusters to fine-grained entities. Experiments on three question-answering benchmarks show that DAG-RAG improves retrieval and answer quality, remains robust as the corpus grows, and benefits from each of its core design components.

AI USE STATEMENT

Generative AI tools were used for language editing and limited assistance with code development and debugging.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the proposed method and training procedure in the main paper, with additional implementation details, hyperparameters, dataset preprocessing, and evaluation protocols provided in the appendix. All experiments are conducted on publicly available datasets following the evaluation settings described in the paper.

ETHICS STATEMENT

All experiments are conducted on publicly available datasets. They do not involve human subjects or personal data.

REFERENCES

Yukun Cao, Zengyi Gao, Zhiyang Li, Xike Xie, S Kevin Zhou, and Jianliang Xu. LEGO-GraphRAG: Modularizing graph-based retrieval-augmented generation for design space exploration. *International Conference on Very Large Databases (VLDB)*, 18(10):3269–3283, 2025.

Boyu Chen, Zirui Guo, Zidan Yang, Yuluo Chen, Junze Chen, Zhenghao Liu, Chuan Shi, and Cheng Yang. PathRAG: Pruning graph-based retrieval augmented generation with relational paths. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 40, pp. 30183–30191, 2026.

Yifan Ding, Qingkai Zeng, and Tim Weninger. ChatEL: Entity linking with chatbots. In *Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING)*, pp. 3086–3097, 2024.

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

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a graph: Encoding graphs for large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

Imane Guellil, Antonio Garcia-Dominguez, Peter R Lewis, Shakeel Hussain, and Geoffrey Smith. Entity linking for english and other languages: a survey. *Knowledge and Information Systems*, 66 (7):3773–3824, 2024.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. LightRAG: Simple and fast retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 10746–10761, 2025.

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

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. From rag to memory: Non-parametric continual learning for large language models. *arXiv preprint arXiv:2502.14802*, 2025.

Yubin Hong, ChaoFan Li, Jingyi Zhang, and Yingxia Shao. Context-aware fine-grained graph RAG for query-focused summarization. In *ACM International Conference on Information and Knowledge Management (CIKM)*, pp. 4802–4806, 2025.

Yuntong Hu, Zhihan Lei, Zheng Zhang, Bo Pan, Chen Ling, and Liang Zhao. GRAG: Graph retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 4145–4157, 2025.

Yiqian Huang, Shiqi Zhang, and Xiaokui Xiao. Ket-rag: A cost-efficient multi-granular indexing framework for graph-rag. In *ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 1003–1012, 2025.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *Transactions on Machine Learning Research*, 2022.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781, 2020.

Meng-Chieh Lee, Qi Zhu, Costas Mavromatis, Zhen Han, Soji Adeshina, Vassilis N Ioannidis, Huzefa Rangwala, and Christos Faloutsos. Hybgrag: Hybrid retrieval-augmented generation on textual and relational knowledge bases. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 879–893, 2025.

Dong Li, Yichen Niu, Ying Ai, Xiang Zou, Biqing Qi, and Jianxing Liu. T-GRAG: A dynamic graphRAG framework for resolving temporal conflicts and redundancy in knowledge retrieval. In *ACM International Multimedia Conference (MM)*, pp. 11880–11889, 2025a.

Mufei Li, Siqi Miao, and Pan Li. Simple is effective: The roles of graphs and large language models in knowledge-graph-based retrieval-augmented generation. In *International Conference on Learning Representations (ICLR)*, 2025b.

Lei Liang, Zhongpu Bo, Zhengke Gui, Zhongshu Zhu, Ling Zhong, Peilong Zhao, Mengshu Sun, Zhiqiang Zhang, Jun Zhou, Wenguang Chen, et al. Kag: Boosting llms in professional domains via knowledge augmented generation. In *Companion Proceedings of the ACM on Web Conference (WWW)*, pp. 334–343, 2025.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

Haoran Luo, E Haihong, Guanting Chen, Yandan Zheng, Xiaobao Wu, Yikai Guo, Qika Lin, Yu Feng, Zemin Kuang, Meina Song, et al. HypergraphRAG: Retrieval-augmented generation via hypergraph-structured knowledge representation. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2025a.

Tongxu Luo, Wenyu Du, HanWen Hao, Min Zhang, Hao Yang, and Benyou Wang. E^2-rag: Towards editable efficient rag by editing compressed kv caches. In *Conference on Language Modeling (COLM)*, 2025b.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9802–9822. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.546. URL https://aclanthology.org/2023.acl-long.546/.

Costas Mavromatis and George Karypis. GNN-RAG: Graph neural retrieval for efficient large language model reasoning on knowledge graphs. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 16682–16699, 2025.

Costas Mavromatis, Soji Adeshina, Vassilis N Ioannidis, Zhen Han, Qi Zhu, Ian Robinson, Bryan Thompson, Huzefa Rangwala, and George Karypis. Byokg-rag: Multi-strategy graph retrieval for knowledge graph question answering. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 27869–27886, 2025.

André Melo, Enting Chen, Pavlos Vougiouklis, Chenxin Diao, Shriram Piramanayagam, Ruofei Lai, and Jeff Z Pan. Don't forget the base retriever! a low-resource graph-based retriever for multi-hop question answering. In *Empirical Methods in Natural Language Processing (EMNLP): Industry Track*, pp. 2564–2572, 2025.

Gerard Pons, Besim Bilalli, and Anna Queralt. Knowledge graphs for enhancing large language models in entity disambiguation. In *International Semantic Web Conference*, pp. 162–179. Springer, 2024.

Nils Reimers and Iryna Gurevych. The curse of dense low-dimensional information retrieval for large index sizes. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 605–611, 2021.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel Ni, Heung-Yeung Shum, and Jian Guo. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *International Conference on Learning Representations (ICLR)*, 2024.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. Is ChatGPT good at search? investigating large language models as re-ranking agents. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 14918–14937, 2023.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Shu Wang, Yixiang Fang, Yingli Zhou, Xilin Liu, and Yuchi Ma. ArchRAG: Attributed community-based hierarchical retrieval-augmented generation. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 40, pp. 15868–15876, 2026.

Zhefan Wang, Huanjun Kong, Jie Ying, Wanli Ouyang, and Nanqing Dong. ROGRAG: A robustly optimized graphRAG framework. In *Annual Meeting of the Association for Computational Linguistics (ACL): System Demonstrations*, pp. 604–613, 2025.

Orion Weller, Michael Boratko, Iftekhar Naim, and Jinhyuk Lee. On the theoretical limitations of embedding-based retrieval. In *International Conference on Learning Representations (ICLR)*, 2026.

Junde Wu, Jiayuan Zhu, Yunli Qi, Jingkun Chen, Min Xu, Filippo Menolascina, Yueming Jin, and Vicente Grau. Medical graph RAG: Evidence-based medical large language model via graph retrieval-augmented generation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 28443–28467, 2025.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-Pack: Packed resources for general Chinese embeddings. In *ACM SIGIR Conference*, pp. 641–649, 2024.

Rui Yang, Boming Yang, Xinjie Zhao, Fan Gao, Aosong Feng, Sixun Ouyang, Moritz Blum, Tianwei She, Yuang Jiang, Freddy Lecue, et al. Graphusion: A RAG framework for scientific knowledge graph construction with a global perspective. In *Companion Proceedings of the ACM on Web Conference (WWW)*, pp. 2579–2588, 2025.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2369–2380, 2018.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 1321–1331, 2015.

Fangyuan Zhang, Zhengjun Huang, Yingli Zhou, Qintian Guo, Zhixun Li, Wensheng Luo, Di Jiang, Yixiang Fang, and Xiaofang Zhou. Erarag: Efficient and incremental retrieval augmented generation for growing corpora. *arXiv preprint arXiv:2506.20963*, 2025a.

Nan Zhang, Prafulla Kumar Choubey, Alexander Fabbri, Gabriel Bernadett-Shapiro, Rui Zhang, Prasenjit Mitra, Caiming Xiong, and Chien-Sheng Wu. SiReRAG: Indexing similar and related information for multihop reasoning. In *International Conference on Learning Representations (ICLR)*, 2025b.

Zaiyi Zheng, Song Wang, Zihan Chen, Yaochen Zhu, Yinhan He, Liangjie Hong, Qi Guo, and Jundong Li. CoRAG: Enhancing hybrid retrieval-augmented generation through a cooperative retriever architecture. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 16088–16101, 2025.

Xiangrong Zhu, Yuexiang Xie, Yi Liu, Yaliang Li, and Wei Hu. Knowledge graph-guided retrieval augmented generation. In *The Nations of the Americas Chapter of the Association for Computational Linguistics (NAACL)*, pp. 8912–8924, 2025.

A Implementation Details

This section supplements the method and experiments with the model configuration, index construction, training procedure, and numerical precision used by DAG-RAG.

A.1 Model Stack

We align the extraction model, base embedding model, and answer generator across systems. Each baseline retains its own indexing and retrieval pipeline; DAG-RAG additionally trains its encoder and router.

Answer generation. Answer generation is performed by **DeepSeek V4 Flash** for DAG-RAG and for every baseline. To reduce generation variability, the model is queried with reasoning disabled, i.e. `"thinking": {"type": "disabled"}`, and with greedy decoding. DAG-RAG issues *no* API model call during grounding: the traversal of §3.2 is executed entirely by the local router described below.

Offline index construction. Entity and relation extraction and description generation are performed by API calls to the same **DeepSeek V4 Flash** model used for answer generation. The same extraction model is used across graph-based methods, but each method retains its own extraction schema and construction procedure (§B.3).

Entity and query encoder. Entity and query representations are produced by `Qwen3-Embedding-0.6B` with output dimension $d = 1024$ and ℓ_2 normalization. Cluster representations are normalized centroids in this embedding space, rather than separately encoded text. We use the encoder's full output dimension and apply no Matryoshka (MRL) truncation. The baselines use the same base encoder without task-specific fine-tuning. DAG-RAG additionally fine-tunes all parameters of this encoder contrastively on oracle seed entities and then freezes it (§A.3); the same fine-tuned encoder is used by the *pure vector retrieval* ablation of §4.5.

Router. The scoring backbone of the router is a 3-layer Transformer trained from scratch rather than initialized from a pretrained checkpoint, and it consumes the representations produced by the encoder above. The backbone has model width $d_{\text{model}} = 512$, 8 attention heads, and a feed-forward expansion of $4\times$ (2,048). Together with the routing head it has 16.70M trainable parameters, 14.01M in the scoring backbone and 2.69M in the routing head (score head, recurrent cell, edge network, and the per-level conditioning parameters of §A.3). Candidate scoring uses this local router; query encoding is also performed online, while entity and centroid representations are precomputed.

A.2 Hierarchical Index Configuration

The DAG index is built bottom-up by recursively applying spherical k-means to the ℓ_2-normalized representations of one level (§3.2), implemented as MiniBatch KMeans over the normalized vectors with the resulting centroids renormalized.

The evaluated indices have three levels: $\mathcal{V}^1$ contains 1,024 top-level clusters, $\mathcal{V}^2$ contains intermediate clusters, and $\mathcal{V}^3 = \mathcal{V}$ contains the leaf entities. The intermediate level contains $\text{round}(\sqrt{1024\,|\mathcal{V}|})$ clusters, as reported in Table 4. The labels L0, L1, and L2 in Table 2 refer to these levels in top-down order.

Table 4: Per-level node counts of the hierarchical DAG index. The columns follow the top-down notation of §3.1; n includes the leaf level.

Dataset	Top $\mathcal{V}^1$	Intermediate $\mathcal{V}^2$	Leaves $\mathcal{V}^3$	n
HotpotQA	1,024	8,396	68,836	3
MuSiQue	1,024	8,590	72,066	3
PopQA	1,024	8,085	63,839	3

Multi-parent linking. After primary-parent clustering, each node below the top level considers its 12 nearest centroids at the level above. An additional parent is retained when its cosine similarity is at least $\tau = 0.15$ and lies within $\delta = 0.12$ of the best parent's similarity, with at most three parents per node. The centroid representations are fixed by clustering. Parents have a soft capacity of 32 children. If all admissible parents are at capacity, the node is attached to its best parent. The largest observed child sets contain 83–106 nodes.

Candidate budget. Online grounding uses a per-level candidate budget $K = 1536$ instead of a fixed beam width. Parents are considered in descending order of their retained-path marginals. A parent is admitted only if adding its complete child set keeps the deduplicated frontier within K; otherwise, it is skipped and the traversal continues to the next parent. Child sets are admitted as whole groups, preserving the complete sibling sets used in Eq. (8). The $1{,}024$ top-level nodes are all scored. At the leaf level, the k entities with the largest retained-path marginals form $\mathcal{V}_q$.

Incremental maintenance. The frozen embedding space permits maintenance of the hierarchy without changing the encoder. The update configuration uses an attachment threshold $\tau_{\text{att}} = 0.92$ and a coverage threshold $\tau_{\text{new}} = 0.6$: a new entity above the attachment threshold is linked to its nearest centroid, while an entity below the coverage threshold enters a pending set associated with its nearest centroid. When a pending set becomes sufficiently large, the affected region is reclustered and changed centroids are propagated upward. Deletion removes an entity and its incident parent edges; undersized clusters can be merged with nearby clusters. These operations concern the hierarchy over already extracted entities; processing new source text still requires entity and relation extraction. Multi-parent links can make an update affect several overlapping regions rather than a single disjoint subtree. The router can be reused when the number of levels and input features remain unchanged. To measure maintenance cost, we incrementally update the HotpotQA hierarchy from the 750-question corpus to the $1{,}000$-question corpus of the nested pools in §B.2, instead of rebuilding it from scratch. The complete update takes 21.8 s, excluding LLM-based entity and relation extraction on the added passages. Update-time retrieval accuracy is not evaluated separately in the reported experiments.

A.3 Encoder Fine-Tuning and Router

Task-aware entity encoder. The encoder of §A.1 is fine-tuned with a contrastive objective on the oracle seed entities $\mathcal{V}_q^*$ of the training questions (§C.2), pulling a query towards the entities that reverse-ground to its supporting sentences and away from the remaining entities of the batch, and is then frozen for both index construction and routing. The entity representations the router consumes are precomputed once offline in this fixed embedding space. All encoder parameters are updated during this stage. The contrastive objective uses a temperature of 0.05.

Router architecture. The router jointly encodes the query and the candidates of one level with the 3-layer Transformer of §A.1, taking as input the precomputed candidate representations, the query token states and the query embedding, together with the structural features of the candidate nodes. It does not serialize any textual description of a candidate, so the candidates of a level are scored jointly in one forward pass. A scalar score head reads the flat node score $s_i^{(\ell)}$ from each candidate's contextualized state $h_i^{(\ell)}$; a recurrent cell carries the path state $\widetilde{z}_{ji}^{(\ell)}$ along each retained parent–child edge; and the edge network D_ψ turns that path state into the residual correction of Eq. (8). The output layer of D_ψ is zero-initialized, so at the start of training the router reduces to a softmax over the flat node scores.

One set of weights serves every level of the DAG, and each level owns a learned level embedding and a learned input adapter with bottleneck width 64, applied to the candidate representations before the shared backbone sees them.

Router hyperparameters. The router is trained with $\mathcal{L} = \mathcal{L}_{\text{route}} + \lambda\,\mathcal{L}_{\text{leaf}}$ (Eq. (11)) using leaf weight $\lambda = 0.17$, with a nominal per-level candidate budget $K = 1536$, matching the inference configuration. All 16.70M router parameters, namely the scoring backbone, the score head, the level-conditioning embedding and adapter, the recurrent cell, and the edge network, are trained jointly from scratch; the entity encoder is frozen throughout.

Optimization. Both training stages use AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.98$, weight decay 0.01) under `bfloat16` autocast on a single GPU, with a linear warmup over the first 6% of steps and gradient clipping at global norm 1.0. The base learning rate is 2×10^{-5}, raised for the two groups of parameters that are both newly initialized and responsible for the routing decision itself. The level-conditioning parameters, namely the per-level embedding and the per-level input adapter of the paragraph above, use $\times 10$; the routing head added on top of the scoring backbone, namely the edge-scoring MLP, the recurrent cell, and the learned initialization vectors of the virtual root, uses $\times 20$. The router is trained for 4 epochs with a batch of one index and four queries per step.

Training frontiers. The frontiers on which the router is trained are constructed offline with a parameter-free cosine router rather than with the router being learned. For every retained parent we materialize its complete child set, so that the sibling-wise denominator in Eq. (8) is exact during training. When the cosine router fails to retain a positive node, that node is restored through its highest-scoring parent and all siblings under that parent are added, which keeps every positive reachable while preserving the exact normalization. The resulting procedure is off-policy: the frontiers do not change as the router is updated.

Training data. The contrastive encoder fine-tuning and the router ($\mathcal{L}$, Eq. (11)) are trained on 1,000 QA pairs from HotpotQA and 1,000 from MuSiQue, together with the passage texts that accompany those questions and the index built over them. PopQA contributes no training data. Evaluation indices are built separately from a disjoint question pool (§B.1).

A.4 Numerical Precision

Training and inference run the router's neural scoring under `bfloat16` autocast, while *every scalar and probability path is kept in `fp32`*. Concretely, the score head output $s_i^{(\ell)}$, the residual produced by the edge network D_ψ, the transition logits of Eq. (8), and the forward recursion of Eq. (9) that accumulates path marginals and path states are all computed in `fp32`. Gradient checkpointing is used for encoder fine-tuning but not for router training.

B Datasets and Graph Construction

B.1 Datasets

We evaluate on three QA benchmarks: **HotpotQA**, open-domain two-hop questions over Wikipedia passages; **MuSiQue**, compositional questions that require chaining multiple supporting facts; and **PopQA** (Mallen et al., 2023), single-hop entity-centric questions over Wikipedia.

For each dataset, the evaluation corpus is associated with a pool of $1{,}000$ questions, of which 250 are sampled uniformly without filtering on difficulty or answerability. The main comparison and entity-recall ablations use the same evaluation questions and full evaluation index. HotpotQA and MuSiQue use question-associated passages. For PopQA, we use the passage corpus and question-level gold-passage annotations from the official PopQA data release accompanying HippoRAG 2 (Gutiérrez et al., 2025). The HotpotQA and MuSiQue training and evaluation question pools are drawn from the development splits and are disjoint. PopQA contributes only an evaluation pool from its single released set.

Each system is evaluated once on 250 questions per dataset; the reported results do not include confidence intervals or repeated-run variability.

B.2 Corpus-Scale Construction

The corpus-scale experiment (§4.4) evaluates a fixed question set against four nested corpora constructed from pools of 250, 500, 750, and $1{,}000$ questions. The scale labels 25%, 50%, 75%, and 100% refer to these pool sizes relative to the largest pool, rather than exact fractions of the deduplicated passage count. Each tier is a nested subset of the next, the 100% tier is the corpus of Table 5, and the evaluated questions and their gold answers are identical at every tier. A fresh index, including the DAG hierarchy, is rebuilt per tier.

Table 5: Dataset statistics. Corpus units are Wikipedia passages. PopQA corpus units and question-level gold passages are taken from the official HippoRAG 2 data release; PopQA is used only for evaluation.

Statistic	HotpotQA	MuSiQue	PopQA
Index questions	1,000	1,000	1,000
Questions evaluated	250	250	250
Question split	dev	dev	-
Corpus units	9,769	10,785	8,676
Corpus size (chars)	5.4 M	4.6 M	4.4 M

B.3 Constructed Graph Statistics

Table 6 reports the offline construction output of DAG-RAG and the graph-based baselines under the shared model stack (§A.1).

Table 6: Constructed graph statistics per method. Each method is reported in the element types its own schema produces: "Entities" counts graph nodes and "Relations"/"Triples"/"Propositions" count the relational units each system extracts, MS GraphRAG additionally reports community summaries, and KAG, whose schema is heterogeneous, separates KnowledgeUnit, entity, SemanticConcept and chunk nodes and records source edges that link extracted content back to its originating chunks. Vanilla RAG builds no graph and is therefore omitted.

Method	Element	HotpotQA	MuSiQue	PopQA
DAG-RAG	Entities	68,836	72,066	63,839
	Relations	67,546	69,887	66,752
LightRAG	Entities	65,670	65,300	64,235
	Relations	88,211	84,249	85,903
SiReRAG	Entities	70,054	67,533	82,572
	Propositions	99,212	103,581	127,843
HippoRAG	Entities	91,065	102,844	98,831
	Triples	128,360	157,830	131,984
MS GraphRAG	Entities	53,617	55,348	45,310
	Triples	53,550	63,631	56,883
	Communities	8,163	7,251	7,999
KAG	KnowledgeUnit nodes	71,196	68,150	80,307
	Entity nodes	141,009	156,151	151,777
	SemanticConcept nodes	53,749	48,151	45,563
	Chunk nodes	8,371	9,974	9,733
	Source edges	427,258	410,085	512,216
	Triples	83,839	81,749	73,921

In KAG's block, entity nodes exclude the *KnowledgeUnit* nodes listed above them, and source edges are provenance links rather than facts. These statistics are produced with KAG version 0.8.0 (§E).

C Metrics and Oracle Gold Entities

C.1 Evidence Granularity

Grounding quality is reported at two granularities.

Passage-level recall is used for the corpus-scale experiment (§4.4): for each question we take the top 20 retrieved passages and report the fraction of the question's gold supporting passages that appear in them, averaged over questions. For HotpotQA and MuSiQue the evidence unit is a supporting

passage, identified from the benchmark's supporting-fact or supporting-paragraph annotations. For PopQA, the gold set is the set of passages assigned to each question in the official HippoRAG 2 release (§B.1). This metric is computed identically for DAG-RAG and every baseline, whatever intermediate representation each system grounds to.

Entity-level recall is used for the router ablations (§4.5): for a retrieval budget of k seed entities, the k of Eq. (3), we report the fraction of the oracle gold entities $\mathcal{V}_q^*$ (§C.2) that the grounded set contains, swept over a range of budgets. It is defined only for the methods that share DAG-RAG's entity index.

The seed and expanded-subgraph sizes are different quantities: the k of Eq. (3) is the number of grounded seed entities, whereas the k of Eq. (4) is the size of the expanded subgraph, which in our pipeline follows from the one-hop expansion rather than being set to the seed budget.

C.2 Oracle Gold-Entity Construction

Across all three datasets, the gold-entity set $\mathcal{V}_q^*$ used by the entity-level recall reported in §4.5 and by the routing supervision $\mathcal{G}_j^{(\ell)}$ of Eq. (10) is obtained by *reverse-grounding* each question's gold supporting evidence and gold answer back to graph entities. PopQA uses the same entity-linking procedure as HotpotQA and MuSiQue, with its question-level gold passages supplied by the HippoRAG 2 release. Each linked entity is assigned a confidence κ through a tiered matcher:

- **Tier 1 — exact match** ($\kappa = 1.0$). *(a) Provenance:* an entity is accepted if the extraction source it records, a (title, sentence index) pair, coincides with one of the question's supporting sentences. *(b) Alias match:* an entity is accepted if its normalized name or alias (at least three characters) occurs as a whole word in a supporting sentence. The gold answer string (excluding `yes`/`no`) is matched by exact alias match ($\kappa = 1.0$), falling back to a nearest-neighbor embedding match above a similarity threshold ($\kappa = 0.7$).
- **Tier 2 — embedding backfill** ($\kappa = 0.7$). Each supporting sentence is encoded and linked to its most similar graph entity when the similarity exceeds `link_sim_threshold`.
- **Tier 3 — LLM arbitration** ($\kappa = 0.5$, optional). If too few entities are linked, fewer than $\max(2, \#\text{supporting sentences})$, the supporting sentences and candidate entities are passed to an LLM that decides which entities are genuinely mentioned. The *evaluation* oracle is built without this LLM step, so every reported entity-level recall figure uses Tier 1 and Tier 2 only.

These matches define a constructed supervision set rather than a benchmark-provided entity annotation. The final $\mathcal{V}_q^*$ is the deduplicated set of linked entity ids, and a per-question weight w equal to the mean κ of its entities is used only as a training confidence weight, not in any evaluation metric.

C.3 Routing Score Decomposition

Table 2 compares gold and non-gold candidates at each routing level. Fix a dataset and a level ℓ, and let $\mathcal{A}^{(\ell)}$ collect every candidate scored at that level over all evaluation questions, with a candidate understood as a query–node occurrence rather than a globally unique node. We split it into a gold part $\mathcal{A}_+^{(\ell)}$ and a non-gold part $\mathcal{A}_-^{(\ell)}$ by the same criterion used for routing supervision (§3.3): a non-leaf candidate is gold if at least one entity of the oracle seed set $\mathcal{V}_q^*$ lies among its descendants; at the leaf level, a candidate is gold if it belongs to $\mathcal{V}_q^*$. Let $s_i^{(\ell)}$ and $\Delta_i^{(\ell)}$ denote the flat score and correction of candidate node i.

The correction gap $\mu^{(\ell)}$ is the mean correction for gold candidates minus that for non-gold candidates,

$$\mu^{(\ell)} = \frac{1}{|\mathcal{A}_+^{(\ell)}|} \sum_{i \in \mathcal{A}_+^{(\ell)}} \Delta_i^{(\ell)} - \frac{1}{|\mathcal{A}_-^{(\ell)}|} \sum_{i \in \mathcal{A}_-^{(\ell)}} \Delta_i^{(\ell)}, \tag{12}$$

and the reported dispersion $\sigma^{(\ell)}$ combines the two within-group standard deviations in quadrature,

$$\sigma^{(\ell)} = \sqrt{\left(\sigma_+^{(\ell)}\right)^2 + \left(\sigma_-^{(\ell)}\right)^2}, \qquad \left(\sigma_\pm^{(\ell)}\right)^2 = \frac{1}{|\mathcal{A}_\pm^{(\ell)}|} \sum_{i \in \mathcal{A}_\pm^{(\ell)}} \left(\Delta_i^{(\ell)} - \overline{\Delta}_\pm^{(\ell)}\right)^2, \tag{13}$$

where $\overline{\Delta}_{\pm}^{(\ell)}$ is the corresponding group mean.

The two flat-score columns report the group means $\overline{s}_{+}^{(\ell)}$ and $\overline{s}_{-}^{(\ell)}$ separately. Kendall's τ is computed over the gold candidates only, between $s_i^{(\ell)}$ and $\Delta_i^{(\ell)}$.

C.4 LLM JUDGE SCORE (LJS) PROMPT

LJS is a binary correctness score assigned by an LLM grader that compares the generated answer against the reference answer, accepting surface-form variation but rejecting different values, missing answers, or abstentions. The exact grader prompt is shown below.

```
You are a strict grader for open-domain question answering. You are given
     a question, the reference (gold) answer, and a candidate answer.
    Decide whether the candidate answer is CORRECT: it must convey the
    same factual answer as the gold answer. Accept differences in surface
     form -- aliases, casing, extra explanatory words, equivalent date/
    number formats, or a fuller phrasing that still contains the gold
    answer. Mark INCORRECT if the candidate gives a different entity/
    value, is missing the answer, hedges, or refuses/abstains. Respond
    with exactly one word on the first line: CORRECT or INCORRECT.

Question: {q}
Gold answer: {gold}
Candidate answer: {pred}
Is the candidate answer correct? Answer CORRECT or INCORRECT.
```

Listing 1: LLM Judge (LJS) grading prompt.

D PROMPTS

This section lists the DAG-RAG construction and shared answer-generation prompts. The evaluation prompt is given in §C.4.

D.1 GRAPH CONSTRUCTION PROMPT

During offline construction (§A.1), the extraction model is prompted per paragraph, through the same API, to emit entities and relational triples as strict JSON. The prompt requests canonical entity names, includes the page-title entity, and asks the model to preserve numeric and attribute facts as triples.

```
You are an information-extraction engine that builds a knowledge graph
    from an encyclopedia paragraph. Extract the salient named entities
    and the factual relations between them. Be precise and faithful to
    the text.
Title: {title}
Sentences (indexed):
{sents}
Return STRICT JSON with this schema (no prose):
{
  "entities": [{"name": "<canonical entity name>", "summary": "<=40-word
    description that PRESERVES specific facts: numbers, dates, capacities
    , populations, quantities, locations>"}],
  "triples": [{"head": "<entity name>", "relation": "<short verb phrase
    >", "tail": "<entity name OR a specific value e.g. a number/date/
    place>", "sent": <sentence index int>}]
}
Rules:
- The page's title entity MUST appear in entities.
- Only use entity names that you list in "entities" (values like numbers/
    dates may appear as triple tails without being listed).
- Keep entity names as they appear (proper nouns); merge obvious pronoun
    references to the named entity.
```

```
- IMPORTANT: capture attribute/numeric facts as triples too, e.g. (X)--
    population-->(35,124), (Arena)--seats-->(3,677), (Y)--founded
    -->(1962). Do not drop the exact figures.
- 3-12 entities, 3-18 triples.
```

Listing 2: Entity/relation extraction prompt used for graph construction.

D.2 Answer-Generation Prompt

Given the retrieved evidence, the generator is instructed to answer using only the provided passages and relations, to reason across multiple evidence units, and to abstain with a fixed string when the evidence is insufficient.

```
You answer questions using ONLY the provided evidence: source passages
    and the relations connecting their entities. Reason over multiple
    passages and relations as needed. If the evidence lacks what is
    required, reply exactly: 'I cannot determine the answer from the
    provided evidence.' Answer concisely with the final answer only.
```

Listing 3: Answer-generation system prompt.

E Baseline Configurations

The main comparison of §4.2 is end to end: every system runs its own complete pipeline, from query to answer, over the same corpus and question set. We therefore do not transplant DAG-RAG's grounding into the baselines; instead we align the parts of each system that are not under study, namely the corpus, the question set, the extraction model (§A.1), the base entity encoder, and the answer generator and its prompt (§D.2), and retain method-specific indexing and retrieval subject to the configurations below. Where a baseline exposes several retrieval modes, we select the mode that exercises entity-level grounding.

Baseline tuning. Baselines use their released retrieval configurations, subject to the shared-model substitutions and method-specific settings below, with no task-specific training: none of them fine-tunes an encoder or trains a router on our training questions. DAG-RAG does both (§A.3). The ablation *pure vector retrieval* uses DAG-RAG's fine-tuned encoder and entity index with cosine grounding instead of the router.

LightRAG. We use the *hybrid* retrieval mode, which combines low-level (entity-centric) and high-level (topic-centric) retrieval.

MS GraphRAG. We use local search, which grounds queries to entities before expanding the retrieved context, rather than global search over community summaries.

KAG. We use KAG version $0.8.0$ and disable reasoning only during final answer generation, using the shared generator and prompt (§D.2). KAG's entity grounding and retrieval procedure are retained. In an observed run, a single KAG question took roughly five hours and consumed about 500k tokens.

SiReRAG. We use its default configuration.

HippoRAG. We distinguish the original HippoRAG (Gutiérrez et al., 2024) and the later HippoRAG 2 (Gutiérrez et al., 2025), which is a separate system. The HippoRAG baseline in all experiments uses the *original* HippoRAG, and we write simply "HippoRAG" throughout; using the HippoRAG 2 PopQA data release does not change the version of the evaluated baseline.

Vanilla RAG. The vector-only baseline builds no graph. The corpus is split into passages, each passage is embedded once with the base encoder of §A.1, and the top-ranked passages under cosine similarity to the query embedding are concatenated into the generator prompt, with the same generator and passage budget as every other system.

F Construction and Inference Cost

We do not report offline construction token counts or training cost. Building the DAG hierarchy from frozen embeddings requires no additional LLM calls beyond entity/relation extraction, but clustering, embedding computation, and encoder/router training still incur computational cost.

F.1 Online Grounding Cost

Grounding requires query encoding, batched node scoring, edge scoring, and retained-path accumulation. Under the candidate-budget rule of §A.2, an n-level index requires at most Kn candidate-scoring instances. For the reported three-level configuration, this gives $1{,}024 + 1{,}536 + 1{,}536 = 4{,}096$ instances per query, approximately 16.8 times fewer than the $68{,}836$ entities in the HotpotQA index. This reduction concerns candidate-scoring instances rather than end-to-end runtime.

DAG-RAG makes no API LLM calls during grounding, so its online LLM tokens are used for answer generation. The main table reports complete-pipeline time, while Table 3 reports retrieval time.

G Effect Sizes and Grounding Headroom

Table 1 shows that DAG-RAG achieves the highest EM on MuSiQue and PopQA, and the highest LJS on all three datasets. It does not lead every metric: KAG has higher EM on HotpotQA (0.572 versus 0.544) and higher F1 on MuSiQue (0.490 versus 0.436). DAG-RAG improves EM over Vanilla RAG by 9.6, 16.0, and 5.6 percentage points on HotpotQA, MuSiQue, and PopQA, respectively. Its end-to-end latency is the lowest among the graph systems on HotpotQA and second-lowest on MuSiQue and PopQA. DAG-RAG has the second-lowest token cost among the graph systems on all three datasets, with token counts close to those of Vanilla RAG.

The shared-encoder comparison is more specific to routing. Relative to pure vector entity retrieval, DAG-RAG improves EM by 4.4, 8.4, and 4.0 points on the three datasets (Table 3). Replacing routed seeds with the constructed oracle entities improves EM by a further 2.0, 2.0, and 4.0 points. These are observed gains under one oracle construction and a fixed downstream pipeline, not a formal upper bound on all possible grounding strategies.

H Limitations

First, the hierarchical, multi-parent index structure is not itself a novel construction: multi-granularity clustering indices and hierarchical summaries have been used in prior retrieval and GraphRAG systems. The contribution of DAG-RAG lies not in the shape of the index but in how grounding is performed over it, namely the path-conditioned router and the budgeted multi-path level-wise traversal that together turn a generic hierarchy into an accurate entity-grounding mechanism.

Second, training the encoder and the router depends on oracle gold-entity supervision obtained by reverse-grounding gold supporting sentences (§C.2). Supporting-fact annotations can support this construction, but entity linking introduces additional noise, and obtaining equivalent supervision may be costly in other domains.

Third, the router is trained on frontiers produced by a fixed cosine router rather than by itself (§A.3). This off-policy choice keeps optimization stable and cheap, but leaves a mismatch between the frontiers seen during training and the model-induced frontiers encountered at inference time, which we do not correct for.

Fourth, extraction errors and omissions limit the evidence available to the router. We do not quantify extraction coverage, cluster distortion, or update-time retrieval quality, so the experiments do not establish robustness to these sources of index change.

Fifth, PopQA tests transfer without PopQA training data, but remains an English, Wikipedia-based benchmark. Transfer to other domains and languages remains untested.

Finally, the main comparison evaluates complete pipelines with unequal task-specific training. Shared-encoder ablations provide a more controlled comparison of routing, but the 250-question, single-run evaluation and the absence of training-cost measurements limit conclusions about statistical reliability and total computational efficiency.