HyperPathsRAG: Multi-Target Path Approximation over Hypergraph Knowledge Graphs for Multi-Hop QA

Guanhua Zhu[1] **Zirui He**[1] **Mengnan Du**[2] **Chenxi Yuan**[1*]
[1]New Jersey Institute of Technology [2]The Chinese University of Hong Kong, Shenzhen
{gz72, zh296, chenxi.yuan}@njit.edu mengnandu@cuhk.edu.cn

Abstract

Multi-hop question answering needs evidence that is scattered across many sentences and ordered along a reasoning chain. Existing RAG systems struggle with this setting because flat chunk retrieval can miss bridge facts, binary graph structures lose n-ary information, and unordered top-K contexts leave evidence composition to the answer model. We introduce **HyperPathsRAG**, a hypergraph-based RAG framework that aligns query decomposition with the corpus index. It decomposes questions into n-ary sub-questions aligned with hypergraph facts, jointly matches all sub-questions to retrieve reasoning paths, scores paths by chain completeness, and presents the retrieved evidence in reasoning order. We evaluate in the *candidate-passage* setting on the 1k-question hard subsets of HotpotQA, 2Wiki, and MuSiQue, with three answering LLMs. HyperPathsRAG attains the best average Exact Match on every backbone (up to 42.2 EM / 56.2 F_1 with Llama-3.3-70B-Instruct), and the best average F_1 on two of three backbones; ablations attribute the bulk of the gain to the hypergraph index and the multi-target retriever. Our code is available at https://github.com/MrFantasyZ/HyperPathsRAG.

1 Introduction

Open domain multi-hop question answering (MHQA) requires a system to connect evidence that is often distributed across different passages (Yang et al., 2018; Ho et al., 2020; Trivedi et al., 2022). This makes retrieval harder than finding passages individually similar to the input question: the system must recover bridging entities and preserve the order in which evidence should be composed. However, most retrieval-augmented generation (RAG) systems retrieve a flat top-K set of text chunks for a large language model (LLM), which is problematic for MHQA in two ways. Bridging passages may have low surface similarity to the original question and therefore fall below the retrieval cutoff (Khattab et al., 2021; Trivedi et al., 2023). Even when relevant passages are retrieved, a flat context provides little structural signal for composing them into a reasoning chain.

Recent work has made retrieval more structured from two directions. On the corpus side, graph-based RAG indexes entities and relations and retrieves subgraphs or paths from query entities, providing entity-level anchors and more explicit reasoning skeletons (Edge et al., 2024; Gutiérrez et al., 2024, 2025; Zhu et al., 2025). Hypergraph RAG further replaces binary edges with hyperedges, allowing one relation node to connect all entities participating in an n-ary fact (Luo et al., 2025; Feng et al., 2026; Hu et al., 2026; Wang et al., 2026a). On the query side, decomposition methods break a complex question into simpler sub-questions, which can expose intermediate entities and make retrieval more targeted (Press et al., 2023; Trivedi et al., 2023; Gong et al., 2026; Wang et al., 2026b). However, these two directions remain misaligned: existing hypergraph RAG methods preserve n-ary facts in the index but still often retrieve from a single query representation, while decomposition methods typically produce binary or triple-style sub-questions that do not match n-ary hyperedges. As a result, current systems either preserve richer facts without fully exploiting the question structure, or decompose the question in a way that reintroduces n-ary information loss.

We propose **HyperPathsRAG**, a hypergraph-based RAG framework for multi-hop question answering. HyperPathsRAG aligns query decomposition with the corpus index: it builds a hypergraph KG whose relation nodes connect all entities participating in a fact, decomposes the input question into n-ary sub-questions aligned with those hyperedges, and retrieves reasoning paths by jointly matching all sub-questions rather than a single query vector.

* Corresponding author.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 32558–32583
October 24-29, 2026 ©2026 Association for Computational Linguistics

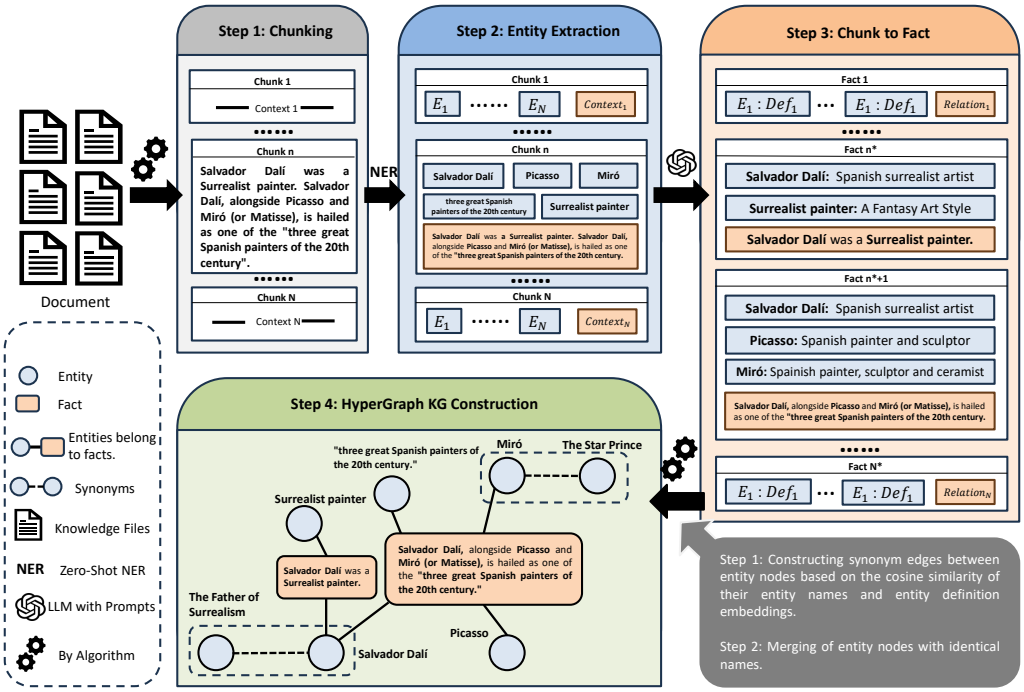

Figure 1: Hypergraph KG construction. **Step 1 (Chunking)** splits each document into non-overlapping chunks. **Step 2 (Entity Extraction)** runs zero-shot NER followed by an LLM refinement call that filters generic candidates, assigns type attributes, and writes short definitions for kept entities. **Step 3 (Chunk-to-Fact)** prompts the LLM to rewrite entity references to canonical names and decouple the chunk into minimal independent facts; each fact becomes a relation node attached to all participating entities by a single *hyperedge*, preserving n-ary structure. **Step 4 (Hypergraph KG Construction)** merges entity nodes of identical normalized names and inserts *synonym edges* between near-duplicates using name and definition embeddings.

Candidate paths are scored by their coverage of the decomposed reasoning chain, and the selected paths are linearized in reasoning order, yielding structured per-path context for the answer model rather than an unordered top-K evidence set.

We evaluate HyperPathsRAG in the *candidate-passage* setting on HotpotQA, 2WikiMultiHopQA, and MuSiQue. Across three answering LLMs, HyperPathsRAG achieves the best average Exact Match among chunk-based, graph-based, and hypergraph-based RAG baselines. The results indicate that multi-hop retrieval benefits from treating evidence as ordered hypergraph paths rather than as independent chunks or unordered subgraphs.

Our contributions are threefold: (1) We propose **HyperPathsRAG**, which aligns n-ary question decomposition with hypergraph facts and retrieves reasoning paths by jointly matching all sub-questions. (2) We introduce path-level scoring and context construction to favor complete, ordered evidence chains. (3) We evaluate on three MHQA benchmarks with three answering LLMs, showing consistent gains over chunk-, graph-, and hypergraph-based RAG baselines, with ablations highlighting the importance of the hypergraph index and multi-target retrieval.

2 Related Work

Retrieval-augmented generation. RAG methods retrieve external evidence and condition generation on the retrieved context (Guu et al., 2020; Karpukhin et al., 2020; Lewis et al., 2021; Izacard and Grave, 2021; Wang et al., 2024). Standard dense retrieval pipelines split the corpus into text chunks and retrieve the top-K passages by similarity, with later work scaling retrieval to larger corpora or making retrieval more adaptive (Izacard et al., 2023; Ram et al., 2023; Asai et al., 2024). For MHQA, however, flat chunk retrieval can miss bridging evidence and does not provide an explicit structure for composing multiple facts.

Graph and hypergraph RAG. Graph-based RAG builds a structured index over entities and relations, allowing retrieval to follow graph neighborhoods, paths, or communities (Edge et al., 2024;

Gutiérrez et al., 2024, 2025; Fan et al., 2025; Zhu et al., 2025; Ao et al., 2025; Huang et al., 2025; Luo et al., 2026; Zhuang et al., 2026; Guo et al., 2025; Zhao et al., 2025; Shen and Kong, 2024; Maftouni et al., 2023). These methods improve over chunk-only retrieval by providing entity-level anchors and structured evidence. Hypergraph RAG further replaces binary edges with hyperedges, allowing a single relation node to connect multiple participating entities (Luo et al., 2025; Feng et al., 2026; Hu et al., 2026; Wang et al., 2026a). Our work follows this direction but differs in pairing the hypergraph index with n-ary query decomposition and path-level multi-target retrieval over the same structure.

Query decomposition for multi-hop retrieval. Query decomposition methods break a complex question into simpler sub-questions before or during retrieval, building on chain-of-thought, task decomposition, and reasoning–tool interaction (Wei et al., 2023; Khot et al., 2022; Yao et al., 2022). Self-Ask and IRCoT show that decomposed retrieval can recover intermediate entities and improve per-hop precision (Press et al., 2023; Trivedi et al., 2023), while later work refines sub-question schemas or iterates retrieval with LLM feedback (Gong et al., 2026; Wang et al., 2026b; Sorokin et al., 2026; Hui et al., 2026; Georgiev et al., 2026). HyperPathsRAG differs by decomposing questions into n-ary sub-questions that are structurally aligned with hypergraph facts, rather than binary or triple-style retrieval targets.

3 HyperPathsRAG

3.1 Hypergraph Knowledge Graph Construction

Figure 1 shows the four-step pipeline that turns a raw corpus into a hypergraph KG with entity nodes, relation (fact) nodes, n-ary hyperedges, and inter-entity synonym edges.

Step 1: Chunking. We split each document into non-overlapping chunks of at most 5 sentences or 800 characters. This keeps each chunk's events strongly co-occurring (simplifying within-chunk coreference) and the extraction prompt short.

Step 2: Entity extraction and refinement. We first run a high-recall NER model, NuNER-Zero (Bogdanov et al., 2024), on each chunk to obtain candidate entities. We then use an LLM-based entity-refinement call to filter generic or over-extracted candidates, assign each kept entity a type attribute, and write a short context-grounded definition. The type attributes are later used to constrain virtual-entity binding during retrieval (§3.2), while the definitions help disambiguate entities with the same surface form. Details of the refinement prompt, controlled vocabulary, and filtering examples are given in Appendix D.1.

Step 3: Chunk-to-fact conversion. Given the refined entity list and the original chunk, LLM Call B first rewrites entity references to their canonical names and then decomposes the chunk into minimal self-contained facts. Each fact becomes a relation node connected by a single hyperedge to all entities it mentions. This preserves n-ary facts as atomic retrieval units, rather than projecting them into multiple binary triples. We separate entity refinement and fact extraction so that each LLM call has a focused output schema. Details of the Call B prompt and reference-rewriting examples are given in Appendix D.2 and Appendix D.3.

Step 4: Hypergraph KG construction. We construct the final hypergraph from the refined entities and extracted facts. Entities with identical normalized names are merged, while near-duplicate entities are connected by undirected synonym edges instead of being collapsed. This lets retrieval expand from one surface form to related aliases without merging distinct entities that share a name. The final graph is $G = (V_e \cup V_r,\ E_h \cup E_s)$, where V_e are entity nodes, V_r are relation nodes, E_h are n-ary fact hyperedges, and E_s are synonym edges. Details of the synonym criterion and embedding construction are given in Appendix D.3.

3.2 Retrieval over the Hypergraph KG

In Figure 2, we summarize the five-stage retrieval pipeline. Given a query, HyperPathsRAG decomposes it into n-ary sub-questions, organizes them into a skeleton DAG, searches the hypergraph to instantiate candidate reasoning paths, scores the paths by chain completeness, and linearizes the top paths as ordered context.

Stage 1: N-ary query decomposition. Given a query Q, an LLM decomposes it into n-ary sub-questions

$$q_i = \big(\texttt{relation},\ \{e_1, \ldots, e_k\}\big),$$

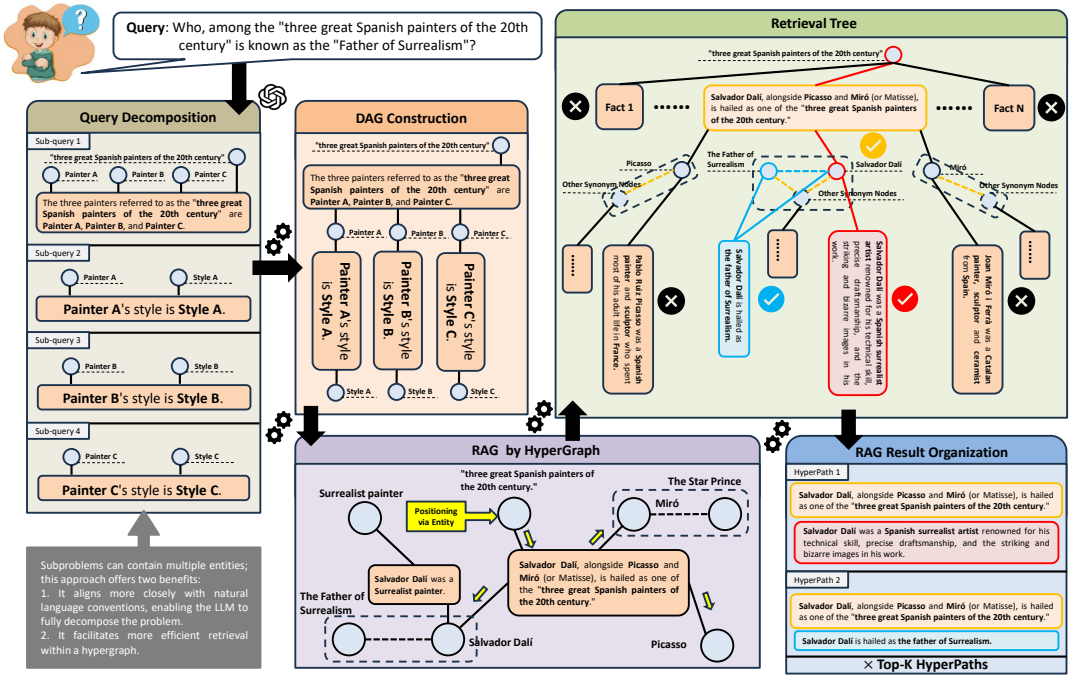

Figure 2: The retrieval pipeline. An LLM decomposes the multi-hop query into a list of n-ary sub-questions (top-left, each carrying one or more entity placeholders). Sub-questions sharing entities are joined into a *skeleton DAG*; real-entity sub-questions become anchor nodes. The hypergraph KG is then searched *multi-target*: every retrieved fact is scored against every sub-question and locked to its best match (lock-and-slide), producing a *retrieval tree* (top-right) whose paths instantiate the skeleton DAG with concrete relations. Each instantiated path is one *HyperPath*; the top-K HyperPaths are emitted as separate paragraphs that preserve the intra-chain reasoning order, then concatenated as the final RAG context (bottom-right).

where `relation` is a complete sentence and each entity slot has an attribute and a `virtual` flag indicating whether it is mentioned in the query or must be retrieved. Unlike binary or triple-style decomposers, the arity k is variable, allowing the query side to match the n-ary facts preserved in the hypergraph. The decomposition prompt, JSON schema, and examples are given in Appendix D.4.

Stage 2: Skeleton DAG construction. We connect sub-questions that share entity slots and orient the resulting structure into a skeleton DAG. Sub-questions containing real query entities serve as anchors, while virtual slots indicate the bridging entities to be discovered. The skeleton DAG therefore defines the abstract reasoning schema that retrieval must instantiate with concrete KG entities and relation nodes. This prevents the decomposed sub-questions from being treated as an unordered set and specifies which discovered entities should unlock the next retrieval hops. Matching rules, deterministic tie-breaking, and cycle handling are described in Appendix E.

Stage 3: Multi-target path search. Starting from the anchor sub-questions, the retriever grows a tree of candidate bindings over the hypergraph. At each step, relation nodes adjacent to the current anchor entities are scored against the decomposed sub-questions, so a relation discovered while expanding one branch can be assigned to the sub-question it best explains. This lock-and-slide behavior prevents early anchor choices from permanently fixing a relation to the branch that happened to retrieve it. We also allow the same relation to be locked by multiple ancestor variants when it supports several sub-questions, which is important for n-ary facts that simultaneously reveal a bridge entity and satisfy another hop. Newly matched relations bind virtual entity slots and spawn child variants for downstream sub-questions, producing a retrieval tree whose root-to-leaf paths instantiate candidate reasoning chains. This multi-target search avoids collapsing the question into a single embedding and allows bridge facts to be recovered from later-hop neighborhoods. Candidate bindings are screened by a *soft* entity-type compatibility filter (Appendix C). Full pseudocode, terminology,

Baseline	Retrieval strategy
No retrieval and chunk retrieval	
NaiveGeneration	Answers directly from the LLM without retrieval.
StandardRAG (Lewis et al., 2021)	Retrieves dense top-K chunks and concatenates them as context.
Graph-based RAG	
GraphRAG (Edge et al., 2024)	Uses community summaries over a hierarchical KG.
HippoRAG 2 (Gutiérrez et al., 2025)	Personalized PageRank over a binary KG with memory reranking.
T2RAG (Gong et al., 2026)	Uses query decomposition with triple-level reasoning.
LogicRAG (Chen et al., 2026)	Builds adaptive reasoning structures without a pre-built graph.
Hypergraph-based RAG	
HyperGraphRAG (Luo et al., 2025)	Extracts n-ary facts and retrieves entity-anchored hypergraph subgraphs.
Hyper-RAG (Feng et al., 2026)	Retrieves over clustered hyperedges in a hypergraph KG.
Cog-RAG (Hu et al., 2026)	Uses a dual-system hypergraph with community-aware retrieval.
HGRAG (Cross-Granularity) (Wang et al., 2026a)	Uses entity and relation seeds at multiple granularities with NV-Embed-v2 (Lee et al., 2025).
HyperPathsRAG (ours)	Ordered hypergraph paths with multi-target matching.

Table 1: Baselines compared in our main experiments.

hybrid relation scoring, adaptive cutoffs, and complexity are given in Appendices A and B.

Stage 4: Chain-completeness scoring. Each root-to-leaf path in the retrieval tree is a candidate reasoning chain. Let $\bar{s}(v_j)$ be the mean score of the relations selected for variant v_j. We score a path $p = (v_1, \ldots, v_m)$ as

$$\text{score}(p) = \frac{1}{n_{\text{subq}}} \sum_{v_j \in p} \bar{s}(v_j), \qquad (1)$$

where n_{subq} is the total number of decomposed sub-questions. Normalizing by the total sub-question count rather than path length penalizes paths that retrieve locally relevant facts but skip required reasoning hops. Thus, a short but incomplete path is less competitive than a longer path that covers more of the decomposed reasoning chain. Path enumeration, parallel-chain handling, and context-time relation selection are described in Appendix E.

Stage 5: Context construction. The top-ranked paths by Eq. 1 form the retrieved context. For each path, we concatenate its selected relation texts in reasoning order; different paths are kept as separate paragraphs before being passed to the answer LLM. Keeping paths separate avoids mixing evidence from incompatible chains, while repeated relations across paths act as a weak agreement signal. This gives the model an ordered evidence structure rather than a flat list of independently retrieved facts. The answering prompt and prompt-order ablation are provided in Appendix Q.

4 Experiments

In this section, we first test whether ordered hypergraph paths improve multi-hop answering. We then examine the mechanism behind the gains through ablations, and finally analyze the remaining errors and computational cost of path-based retrieval.

4.1 Datasets and Corpus Scope

We evaluate HyperPathsRAG in the *candidate-passage* setting on three MHQA benchmarks: HotpotQA (Yang et al., 2018), 2WikiMultiHopQA (Ho et al., 2020), and MuSiQue (Trivedi et al., 2022). Following recent graph-RAG work (Gutiérrez et al., 2025; Wang et al., 2026a), we use the 1,000-question hard subset of each benchmark.

The KG is built per question from that question's released candidate passages, *not* the full Wikipedia; retrieval searches only that KG. HotpotQA uses only the gold supporting passages, forming an *oracle-document* corpus ("HotpotQA$_{\text{oracle}}$"), while 2WikiMultiHopQA (4+6) and MuSiQue (4+16) use the standard distractor pools. The gold answer is never visible at retrieval. Our results therefore establish effectiveness under controlled candidate pools rather than full-corpus retrieval quality or scalability (see Limitations); index statistics are given in Appendix I.

4.2 Baselines and Protocol

We benchmark against ten baselines spanning no retrieval, chunk retrieval, graph-based RAG, and hypergraph-based RAG. Table 1 summarizes their retrieval strategies. For every method, we rerun evaluation under the same per-question candidate

Method	Answer LLM	HotpotQA$_{\text{oracle}}$		2WikiMultiHopQA		MuSiQue		Average	
		EM	F_1	EM	F_1	EM	F_1	EM	F_1
NaiveGeneration	Qwen3-14B	28.0	40.2	22.4	30.5	5.2	12.4	18.5	27.7
	Llama-3.3-70B-Instr.	32.6	45.1	28.3	37.8	7.4	15.6	22.8	32.8
	GPT-4o-mini	30.8	43.2	26.1	35.2	6.5	13.8	21.1	30.7
StandardRAG	Qwen3-14B	42.5	55.3	30.4	42.1	10.1	20.4	27.7	39.3
	Llama-3.3-70B-Instr.	48.2	62.4	35.6	48.0	12.3	22.6	32.0	44.3
	GPT-4o-mini	46.1	60.1	33.2	45.6	11.4	21.3	30.2	42.3
GraphRAG	Qwen3-14B	38.0	50.1	34.8	47.0	8.2	17.6	27.0	38.2
	Llama-3.3-70B-Instr.	43.5	55.8	40.2	51.7	10.1	19.8	31.3	42.4
	GPT-4o-mini	41.6	53.7	38.1	49.5	9.4	18.7	29.7	40.6
HippoRAG 2	Qwen3-14B	42.1	**56.0**	49.8	**66.4**	15.2	26.4	35.7	**49.6**
	Llama-3.3-70B-Instr.	47.4	61.9	54.7	68.1	17.6	29.8	39.9	53.3
	GPT-4o-mini	45.6	60.0	52.9	66.4	16.9	28.7	38.5	51.7
HyperGraphRAG	Qwen3-14B	38.4	52.1	41.7	55.1	12.3	22.7	30.8	43.3
	Llama-3.3-70B-Instr.	44.0	57.6	46.9	61.5	14.4	25.8	35.1	48.3
	GPT-4o-mini	41.8	55.6	44.6	59.3	13.2	24.0	33.2	46.3
Hyper-RAG	Qwen3-14B	37.2	50.0	39.6	53.4	11.4	21.0	29.4	41.5
	Llama-3.3-70B-Instr.	42.8	55.7	44.9	59.8	13.3	23.6	33.7	46.4
	GPT-4o-mini	40.7	54.0	42.8	57.8	12.1	22.1	31.9	44.6
Cog-RAG	Qwen3-14B	39.2	52.0	42.7	56.1	13.0	23.1	31.6	43.7
	Llama-3.3-70B-Instr.	44.8	57.5	47.9	61.9	15.5	26.7	36.1	48.7
	GPT-4o-mini	43.0	56.0	45.8	59.7	14.3	25.0	34.4	46.9
HGRAG	Qwen3-14B	43.4	**56.0**	50.6	64.0	16.2	27.4	36.7	49.1
	Llama-3.3-70B-Instr.	49.0	63.5	**56.3**	70.4	19.0	**31.9**	41.4	55.3
	GPT-4o-mini	47.2	61.8	54.0	68.0	17.8	30.2	39.7	53.3
T2RAG	Qwen3-14B	40.0	53.2	38.0	49.6	13.0	23.5	30.3	42.1
	Llama-3.3-70B-Instr.	45.8	59.6	43.2	54.7	15.7	26.9	34.9	47.1
	GPT-4o-mini	43.9	57.5	41.3	53.0	14.6	25.7	33.3	45.4
LogicRAG	Qwen3-14B	41.0	54.0	45.2	58.3	14.0	24.8	33.4	45.7
	Llama-3.3-70B-Instr.	46.7	60.7	50.4	63.9	16.7	28.5	37.9	51.0
	GPT-4o-mini	44.9	58.8	48.1	61.7	15.6	27.0	36.2	49.2
HyperPathsRAG (ours)	Qwen3-14B	**45.2**	55.1	**51.1**	64.2	**18.0**	**29.2**	**38.1**	49.5
	Llama-3.3-70B-Instr.	**51.2**	**65.0**	53.6	**72.1**	**21.7**	31.6	**42.2**	**56.2**
	GPT-4o-mini	**49.3**	**63.1**	**56.2**	**69.8**	**20.1**	**32.4**	**41.9**	**55.1**

Table 2: Main results on multi-hop QA benchmarks across three answering LLMs (Qwen3-14B, Llama-3.3-70B-Instruct, GPT-4o-mini). The retriever is held fixed across LLMs; across-LLM gaps in the same row reflect the answer model only. *Every* cell (all baselines and our method) is measured by us under one identical protocol (corpus, candidate pool, answering LLM, prompt, and metric; §4); no externally reported numbers are mixed in. All Average columns are computed across the three benchmarks for each row. Within each LLM block, the best value in every metric column is shown in **bold** (ties: both bolded).

pool, KG-construction budget, answering LLM, decoding configuration, prompt, and metric. This yields a controlled comparison rather than a mixture of externally reported numbers. We report Exact Match (EM) and token-level F_1 under SQuAD-style normalisation (Rajpurkar et al., 2016).

4.3 Implementation Details

We use Qwen3-14B (Yang et al., 2025) for KG construction and query decomposition, served locally with Ollama. Entity lookup uses BM25 (Robertson and Zaragoza, 2009) prefiltering followed by dense reranking, and relation scoring combines dense similarity with a normalized BM25 boost. Entity and relation embeddings are computed with BGE-large-en-v1.5 (Xiao et al., 2024). The multi-target retriever uses fixed hyperparameters across all experiments, including adaptive top-K relation selection and a global variant cap; full values and search details are given in Appendix B.

For answer generation, retrieved paths are passed to one of three answering LLMs: Qwen3-14B, Llama-3.3-70B-Instruct (Grattafiori et al., 2024), and GPT-4o-mini. Within each answering LLM, all methods use the same answer prompt and decoding configuration. KG construction runs with

a 16K context window and prefix caching; tabular chunks detected by regex use a table-aware extraction prompt. Full prompt templates, decoding parameters, runtime settings, dependency versions, and timing breakdowns are provided in Appendix P. Appendices C–N report supporting analyses: entity-type filtering, parallel chains, the path-selection objective, index and decomposition statistics, and EM/F_1 divergences.

4.4 Main Results

Table 2 reports EM and F_1 on the three benchmarks with three answering LLMs: Qwen3-14B (Yang et al., 2025), Llama-3.3-70B-Instruct (Grattafiori et al., 2024), and GPT-4o-mini. The retriever and candidate pool are fixed within each comparison, so across-LLM differences reflect the answer model. We include Llama-3.3-70B-Instruct because the closest prior work, HGRAG (Wang et al., 2026a), uses the same datasets and LLM. All baselines and HyperPathsRAG are re-run under the controlled protocol described above in Section 4.2.

Analysis. Three patterns stand out. First, structured retrieval matters more on harder benchmarks: while most RAG methods are relatively close on $\text{HotpotQA}_{\text{oracle}}$, the spread on MuSiQue is much larger, suggesting that longer compositional questions benefit more from explicit evidence structure. Second, graph and hypergraph methods generally outperform flat chunk retrieval on 2WikiMultiHopQA and MuSiQue, and the strongest baselines are those that also organize evidence for reasoning, such as HGRAG and HippoRAG 2.

Third, HyperPathsRAG achieves the best average EM under all three answering LLMs and the best average F_1 under two of them, trailing HippoRAG 2 by only 0.1 F_1 with Qwen3-14B. Against the strongest baseline, HGRAG, the average EM gains are $+1.4$, $+0.8$, and $+2.2$ under Qwen3-14B, Llama-3.3-70B-Instruct, and GPT-4o-mini. Per dataset, HyperPathsRAG improves $\text{HotpotQA}_{\text{oracle}}$ on every LLM by $+1.8$ to $+2.2$ EM and MuSiQue by $+1.8$ to $+2.7$ EM; it also wins 2WikiMultiHopQA under Qwen3-14B and GPT-4o-mini, but trails HGRAG by 2.7 EM under Llama-3.3-70B-Instruct. Overall, the margins over HGRAG are modest but consistent, supporting the claim that multi-hop QA benefits from treating evidence as ordered hypergraph paths, within the candidate-passage scope of §4.

Variant	EM avg	F_1 avg
HyperPathsRAG (full)	**42.2**	**56.2**
w/o n-ary decomposition	35.5	43.4
w/o multi-target	29.2	41.6
w/o chain-completeness	40.8	53.5
w/o order-preserving ctx	37.5	49.0
w/o hypergraph	23.7	30.3

Table 3: Ablation of HyperPathsRAG components, averaged across HotpotQA, 2Wiki and MuSiQue with Llama-3.3-70B-Instruct. The protocol matches Table 2.

4.5 Ablation and Component Analysis

We ablate the main design choices of HyperPathsRAG on the average of the three datasets using Llama-3.3-70B-Instruct (Table 3). The variants test three parts of the method: the alignment between query and index (**w/o n-ary decomposition**, **w/o hypergraph**), the retrieval objective (**w/o multi-target**), and the path-level organization of evidence (**w/o chain-completeness**, **w/o order-preserving context**). In **w/o n-ary decomposition**, the decomposer is constrained to emit binary or triple-style sub-questions. In **w/o hypergraph**, n-ary relation nodes are replaced with all pairwise binary triples. In **w/o multi-target**, retrieval is driven by a single query representation rather than by jointly matching all sub-questions. The two path-level ablations replace Eq. 1 with a flat mean over selected relations, or reorganize retrieved relations as a flat top-K list or by-sub-question grouping.

Removing the hypergraph index causes the largest drop ($42.2 \rightarrow 23.7$ EM), because projecting each n-ary relation node into pairwise triples fragments joint constraints and inflates node degree. We therefore read this result as an upper bound on the indexing contribution rather than a like-for-like binary-KG comparison. Removing multi-target retrieval is the second largest drop (-13.0 EM), confirming that collapsing all sub-questions into a single query embedding loses per-hop bridge signals. N-ary decomposition also matters (-6.7 EM), suggesting that binary or triple-style query decomposition reintroduces on the query side the same n-ary loss that the hypergraph index is designed to avoid. Finally, order-preserving context (-4.7 EM) and chain-completeness scoring (-1.4 EM) show that retrieved relations must be both complete and presented in reasoning order. These drops should not be interpreted additively, since the components

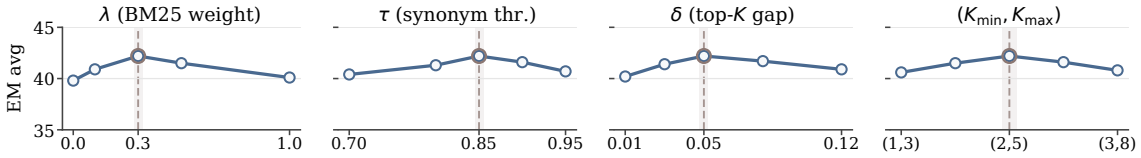

Figure 3: Hyperparameter sensitivity. Each panel varies one retrieval hyperparameter while keeping the others fixed. The vertical dashed line marks the default used in the main experiments. EM remains within a narrow range across all sweeps, suggesting that HyperPathsRAG is not brittle to a narrow hyperparameter setting.

Failure mode	n	**%**
Long-tail relation	**19**	**38**
Decomposition under-coverage	14	28
KG-missing fact	8	16
LLM answering error	5	10
Entity disambiguation	3	6
Annotation noise	1	2
Total	50	100

Table 4: Failure modes from 50 sampled errors, roughly balanced across the three benchmarks. This is a qualitative sample rather than a population-level distribution. n is the raw count and % is the share of the sample.

interact rather than contribute independently.

A prompt-order ablation further supports the role of ordered context. With the same retrieved paths, asking the LLM to produce reasoning before the final answer reaches 42.2 EM, compared with 38.1 when the answer is produced before the reasoning trace and 37.4 with no reasoning trace (Appendix Q). This suggests that ordered path context is most useful when the model is prompted to consume the evidence before committing to an answer. We also test whether the retrieval gains depend on a narrow hyperparameter setting. Figure 3 varies the BM25 weight λ, synonym threshold τ, adaptive top-K gap δ, and $(K_{\min}, K_{\max})$ while keeping the other parameters fixed. EM remains within a narrow range across all sweeps, indicating that HyperPathsRAG is robust to moderate parameter changes rather than being tuned to a fragile point. Full EM/F_1 values and a per-parameter discussion are provided in Appendix J.

4.6 Failure Modes and Efficiency

Failure modes. We first examine where HyperPathsRAG still fails. Table 4 summarizes a qualitative analysis of 50 incorrect predictions, sampled roughly evenly across the three benchmarks. The two largest categories are long-tail relation errors (38%), where the relevant relation is reachable in the KG but ranked below the final path cutoff, and decomposition under-coverage (28%), where the query decomposition omits a needed hop or merges multiple steps into one underspecified sub-question. KG-missing-fact errors account for 16% of the sample, suggesting that the main remaining bottleneck is less about extracting more facts and more about exposing the right chain through better decomposition and relation reranking. The smaller categories point to answer-model mistakes and occasional synonym/disambiguation errors. Appendix L provides per-category examples and discusses likely fixes.

Efficiency. Table 5 reports runtime on the MuSiQue 1k-question pool. HyperPathsRAG is slower than lightweight graph and hypergraph baselines: retrieval takes 5.1 seconds per query, compared with 1.2 seconds for HippoRAG 2 and 0.256 seconds for HyperGraphRAG. The overhead mainly comes from per-query decomposition, variant-tree growth, and path-level scoring, dominated by the decomposition call. However, HyperPathsRAG remains faster than GraphRAG, and its KG construction cost is moderate: higher than HippoRAG 2 and HyperGraphRAG due to the extra NER and two-call refinement pipeline, but lower than GraphRAG. Overall, HyperPathsRAG trades latency for more complete and better organized evidence, making it better suited to offline, batch, or accuracy-sensitive MHQA than low-latency serving. Appendix O gives the full cost–quality discussion, memory usage, and the HGRAG comparison, and Appendix P implementation-level timing.

5 Conclusion

We presented **HyperPathsRAG**, a hypergraph-based RAG framework that aligns n-ary query decomposition with n-ary facts, retrieves evidence by jointly matching all sub-questions, and presents selected relations as ordered path contexts. Across

Method	Retr. (s/q)	KG build (s/1k tok.)
GraphRAG	10.7	9.272
HippoRAG 2	1.2	**2.758**
HyperGraphRAG	**0.256**	3.084
HyperPathsRAG	5.1	4.072

Table 5: Efficiency on the MuSiQue 1k-question pool using an A100 80GB GPU. Lower is better; the best value in each column is bolded.

three MHQA benchmarks and three answering LLMs, HyperPathsRAG achieves the best average EM in the candidate-passage setting, with ablations showing that the hypergraph index and multi-target retrieval contribute most to the gains. These results suggest that MHQA retrieval should return connected evidence chains rather than flat sets of relevant facts.

Limitations

Our evaluation is confined to the *candidate-passage* setting: each question is indexed and searched only within its own released candidate pool, so the results establish effectiveness under controlled pools rather than full-corpus retrieval cost or recall, which remains the most important direction for future work. We report downstream QA metrics and component ablations but no retrieval-level recall, so the claim that the retrieval mechanism itself drives the gains is supported only indirectly (Appendix N). Eq. 1 also scores each path independently, optimising per-path rather than selected-set completeness — a genuine mismatch for comparative and parallel-chain questions, which a set-level marginal-gain objective would address (Appendix G).

Three practical dependencies remain. Retrieval is anchored on extracted entities, and replacing NuNER-Zero with spaCy or GLiNER (Zaratiana et al., 2024) costs 2.2 and 3.3 end-to-end EM, respectively (Appendix R). Retrieval takes 5.1 s per query, 3.8–20× slower than the competitive baselines for average EM gains of +0.8 to +2.2, suiting offline or accuracy-sensitive use rather than interactive serving. And all LLM stages use sampling, so repeated runs can differ; we do not quantify this variance across seeds.

References

Tu Ao, Yanhua Yu, Yuling Wang, Yang Deng, Zirui Guo, Liang Pang, Pinghui Wang, Tat-Seng Chua, Xiao Zhang, and Zhen Cai. 2025. Lightprof: A lightweight reasoning framework for large language model on knowledge graph. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 23424–23432.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avi Sil, and Hannaneh Hajishirzi. 2024. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *International conference on learning representations*, volume 2024, pages 9112–9141.

Sergei Bogdanov, Alexandre Constantin, Timothée Bernard, Benoit Crabbé, and Etienne P Bernard. 2024. Nuner: Entity recognition encoder pre-training via llm-annotated data. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11829–11841.

Shengyuan Chen, Chuang Zhou, Zheng Yuan, Qinggang Zhang, Zeyang Cui, Hao Chen, Yilin Xiao, Jiannong Cao, and Xiao Huang. 2026. You don't need pre-built graphs for rag: Retrieval augmented generation with adaptive reasoning structures. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 30270–30278.

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

Tianyu Fan, Jingyuan Wang, Xubin Ren, and Chao Huang. 2025. Minirag: Towards extremely simple retrieval-augmented generation. *arXiv preprint arXiv:2501.06713*.

Yifan Feng, Hao Hu, Shihui Ying, Xingliang Hou, Shiquan Liu, Mingyuan Yang, Junchang Li, Shaoyi Du, Nanning Zheng, Han Hu, and Yue Gao. 2026. Hyper-rag: Combating llm hallucinations using hypergraph-driven retrieval-augmented generation. *Nature Communications*.

Dobrik Georgiev, Kheeran Naidu, Alberto Cattaneo, Federico Monti, Carlo Luschi, and Daniel Justus. 2026. Ultrag: A universal simple scalable recipe for knowledge graph rag. *arXiv preprint arXiv:2603.28773*.

Shengbo Gong, Xianfeng Tang, Qi He, Carl Yang, and Wei Jin. 2026. Beyond chunks and graphs: Retrieval-augmented generation through triplet-driven thinking. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 26282–26308.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh

Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Yucan Guo, Miao Su, Saiping Guan, Zihao Sun, Xiaolong Jin, Jiafeng Guo, and Xueqi Cheng. 2025. PilotRAG: Teaching LLMs multi-turn hybrid RAG via reinforcement learning.

Bernal J Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in neural information processing systems*, 37:59532–59569.

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

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625.

Hao Hu, Yifan Feng, Ruoxue Li, Rundong Xue, Xingliang Hou, Zhiqiang Tian, Yue Gao, and Shaoyi Du. 2026. Cog-rag: cognitive-inspired dual-hypergraph with theme alignment retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 31032–31040.

Haoyu Huang, Yongfeng Huang, Junjie Yang, Zhenyu Pan, Yongqiang Chen, Kaili Ma, Hongzhi Chen, and James Cheng. 2025. Retrieval-augmented generation with hierarchical knowledge. In *EMNLP (Findings)*, pages 6044–6060.

Yulong Hui, Chao Chen, Zhihang Fu, Yihao Liu, Jieping Ye, and Huanchen Zhang. 2026. Interact-rag: Reason and interact with the corpus, beyond black-box retrieval. In *International Conference on Learning Representations*, volume 2026, pages 112154–112174.

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.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2023. Atlas: Few-shot learning with retrieval augmented language models. *Journal of Machine Learning Research*, 24(251):1–43.

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.

Omar Khattab, Christopher Potts, and Matei Zaharia. 2021. Baleen: Robust multi-hop reasoning at scale via condensed retrieval. *Advances in Neural Information Processing Systems*, 34:27670–27682.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*.

Chankyu Lee, Rajarshi Roy, Mengyao Xu, Jonathan Raiman, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. 2025. Nv-embed: Improved techniques for training llms as generalist embedding models. In *International Conference on Learning Representations*, volume 2025, pages 79310–79333.

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. 2021. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Preprint*, arXiv:2005.11401.

Haoran Luo, Haihong E, Guanting Chen, Yandan Zheng, Xiaobao Wu, Yikai Guo, Qika Lin, Yu Feng, Zemin Kuang, Meina Song, Yifan Zhu, and Luu Anh Tuan. 2025. Hypergraphrag: Retrieval-augmented generation via hypergraph-structured knowledge representation. *Preprint*, arXiv:2503.21322.

Linhao Luo, Zicheng Zhao, Reza Haffari, Dinh Phung, Chen Gong, and Shirui Pan. 2026. Gfm-rag: graph foundation model for retrieval augmented generation. *Advances in Neural Information Processing Systems*, 38:36371–36405.

Maede Maftouni, Bo Shen, Andrew Chung Chee Law, Niloofar Ayoobi Yazdi, Fahimeh Hadavand, Fereshte Ghiasvand, and Zhenyu Kong. 2023. A mask-guided attention deep learning model for covid-19 diagnosis based on an integrated ct scan images database. *IISE Transactions on Healthcare Systems Engineering*, 13(2):132–149.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 2383–2392.

Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. 2023. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and trends® in information retrieval*, 4(1-2):1–174.

Bo Shen and Zhenyu Kong. 2024. Active defect discovery: A human-in-the-loop learning method. *IISE Transactions*, 56(6):638–651.

Artyom Sorokin, Nazar Buzun, Aleksandr Anokhin, Egor Vedernikov, Petr Anokhin, Mikhail Burtsev, and Evgeny Burnaev. 2026. Q-rag: Long context multi-step retrieval via value-based embedder training. In *International Conference on Learning Representations*, volume 2026, pages 32481–32506.

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

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 10014–10037.

Changjian Wang, Weihong Deng, Weili Guan, Quan Lu, and Ning Jiang. 2026a. Cross-granularity hypergraph retrieval-augmented generation for multi-hop question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 33368–33376.

Liang Wang, Haonan Chen, Nan Yang, Xiaolong Huang, Zhicheng Dou, and Furu Wei. 2026b. Chain-of-retrieval augmented generation. *Advances in Neural Information Processing Systems*, 38:59888–59915.

Zhiyuan Wang, Jinhao Duan, Chenxi Yuan, Qingyu Chen, Tianlong Chen, Yue Zhang, Ren Wang, Xiaoshuang Shi, and Kaidi Xu. 2024. Word-sequence entropy: Towards uncertainty estimation in free-form medical question answering applications and beyond. *Preprint*, arXiv:2402.14259.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. 2024. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*, pages 641–649.

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. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Urchade Zaratiana, Nadi Tomeh, Pierre Holat, and Thierry Charnois. 2024. Gliner: Generalist model for named entity recognition using bidirectional transformer. In *Proceedings of the 2024 conference of the north american chapter of the association for computational linguistics: Human language technologies (volume 1: Long papers)*, pages 5364–5376.

Haiyan Zhao, Xuansheng Wu, Fan Yang, Bo Shen, Ninghao Liu, and Mengnan Du. 2025. Denoising concept vectors with sparse autoencoders for improved language model steering. *Preprint*, arXiv:2505.15038.

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.

Luyao Zhuang, Shengyuan Chen, Yilin Xiao, Huachi Zhou, Yujing Zhang, Hao Chen, Qinggang Zhang, and Xiao Huang. 2026. Linearrag: Linear graph retrieval augmented generation on large-scale corpora. In *International Conference on Learning Representations*, volume 2026, pages 147053–147075.

A Extended Terminology

Several terms used in §3.2 are introduced informally in prior work. We fix their meaning here, and state precisely what the hypergraph formulation does and does not claim.

Hypergraph and hyperedge. The index is $G = (V_e \cup V_r,\ E_h \cup E_s)$ with entity nodes V_e, relation (fact) nodes V_r, n-ary fact hyperedges E_h, and synonym edges E_s. A hyperedge $h \in E_h$ attaches one relation node $r \in V_r$ to the entity set $\mathcal{E}(r) = \{e_1, \ldots, e_k\} \subseteq V_e$ participating in that

fact, with variable arity $k = |\mathcal{E}(r)|$. We use "hypergraph" to name the *semantic retrieval unit*: one fact jointly relates a variable-size entity set and is scored and selected *atomically*, rather than being decomposed into independent binary triples. We do *not* claim expressive power beyond a reified bipartite encoding: computationally, the same incidence structure can be stored as a bipartite graph over $V_e \cup V_r$, exactly as n-ary facts are reified as anonymous hub nodes in Freebase CVT nodes or Wikidata statement nodes. What the formulation buys is not representational novelty but alignment: because the fact is one node, a decomposed n-ary sub-question can be matched against it as a whole, which is what Stage 3 exploits.

Sub-question, slot, and variant. A sub-question is $q_i = (\texttt{relation}, \{s_1, \ldots, s_k\})$, where `relation` is a complete sentence and each *slot* s carries a name, a type attribute, and a `virtual` flag (`virtual`=true marks an unknown to be discovered). A *variant* is a partially instantiated retrieval state $v = (q_i,\ \beta,\ A)$, comprising the sub-question q_i it is currently trying to satisfy, a binding β mapping already-resolved virtual slots to concrete KG entities, and an anchor set $A \subseteq V_e$ of entities whose incident relations may be expanded next. Two variants with the same (q_i, β) denote the same retrieval state and are deduplicated (Algorithm 1, line 21). A variant is thus the unit of search-state bookkeeping: it records *which* sub-question is being resolved and *under which* committed bindings.

Retrieval (variant) tree. Expansion produces the *retrieval tree* — equivalently the variant tree — $\mathcal{T}$. Its roots are the variants created for anchor sub-questions, i.e. those containing at least one real query entity. A child variant extends its parent's binding by exactly one newly bound virtual slot and advances to a downstream skeleton sub-question sharing that slot. Consequently every root-to-leaf walk carries a consistent binding chain and visits each sub-question at most once.

HyperPath. A *HyperPath* is a root-to-leaf path $p = (v_1, \ldots, v_m)$ of $\mathcal{T}$ together with the relation nodes locked to its variants, linearised in reasoning order. It is a *retrieval-level object*: an ordered evidence chain over matched n-ary facts produced by the variant-tree search. It is not a new hypergraph-theoretic definition of a path, and no claim in this paper depends on reading it as one.

Algorithm 1 Anchor-Driven Variant Tree (Stage 3)

Require: query Q; hypergraph G; skeleton DAG with sub-questions $\{q_1, \ldots, q_n\}$ and anchor set $\mathcal{A}$
Ensure: variant lineage tree $\mathcal{T}$
1: $\mathcal{T} \leftarrow \varnothing$; queue $\mathcal{Q} \leftarrow [\,]$
2: **for** each anchor sub-question $q_a \in \mathcal{A}$ **do**
3: $E \leftarrow$ BM25-then-dense lookup of q_a's real entities
4: $v_a \leftarrow (qid{=}q_a,\ bind{=}\varnothing,\ anchor{=}E \cup \textsc{SynClosure}(E))$
5: add v_a to $\mathcal{T}$; enqueue v_a
6: **end for**
7: **while** $\mathcal{Q} \neq [\,]$ **and** $|\mathcal{T}| < \textsc{MaxVar}$ **do**
8: pop group $\mathcal{G}$ of variants sharing the same *anchor*
// Step A: ancestor contention with multi-lock
9: $R \leftarrow \bigcup_{e \in anchor(\mathcal{G})} \textsc{Rel}(e) \cup \textsc{ComposedRetrieve}(Q, \mathcal{G})$
10: $\mathcal{C} \leftarrow \bigcup_{v \in \mathcal{G}} \textsc{Ancestors}(v)$ ▷ contestants
11: **for** each contestant $a \in \mathcal{C}$ **do**
12: score every $r \in R$ against q_a by dense–sparse hybrid score
13: $lock(a)$ += $\textsc{AdaptiveTopK}(R; K_{\min}, K_{\max}, \delta_{\text{rel}}, \tau_{\text{rel}})$
▷ no single-winner: r may be locked by several a
14: **end for**
// Step B: virtual binding → spawn children
15: **for** each $v \in \mathcal{G}$ with newly locked relations **do**
16: **for** each unbound virtual ν in v **do**
17: $C \leftarrow$ non-anchor endpoints of v's locked relations, kept if $\textsc{AttrCompat}(\cdot, \nu)$ ▷ soft type pre-filter: keep if compatible *or* untyped; see App. C
18: $B \leftarrow \textsc{AdaptiveTopK}(C; \cdot, N_{\max}, \delta_{\text{bind}}, \tau_{\text{bind}})$ by avg. name/name+def sim
19: **for** each $e \in B$ **do**
20: $q' \leftarrow$ next skeleton sub-question sharing ν
21: $v' \leftarrow (q',\ bind(v) \cup \{\nu{\to}e\},\ \{e\} \cup \textsc{SynClosure}(e))$
22: **if** $(q', bind(v')) \notin \mathcal{T}$ **then** add v' to $\mathcal{T}$; enqueue v'
23: **end if**
24: **end for**
25: **end for**
26: **end for**
27: **end while**
28: **return** $\mathcal{T}$

B Retrieval Algorithm and Complexity

Algorithm 1 gives the explicit pseudocode for the multi-target retrieval of Stage 3 (§3.2); it produces the variant lineage tree that Stage 4 then scores. All constants are the defaults used in every experiment: $K_{\min}$=2, $K_{\max}$=5, δ_{rel}=0.05, τ_{rel}=0.5, δ_{bind}=0.05, τ_{bind}=0.5, $N_{\max}$=5, and a global cap $\textsc{MaxVar}$=600 on the number of variants.

Complexity. Let n be the number of sub-questions, d the average degree of an anchor entity (size of $\textsc{Rel}(e)$), and L the skeleton-DAG depth. Each group runs Step A over a pool of $O(d)$ relations against $O(L)$ ancestors, i.e. $O(dL)$ scor-

ing operations; binding spawns at most $N_{\max}$ children per virtual. The variant count is bounded by the explicit cap MAXVAR, so the whole loop is $O(\text{MAXVAR} \cdot dL)$ in the worst case; in practice the loop converges in 3–5 iterations and the variant count stays in the low hundreds per query (App. P). COMPOSEDRETRIEVE is a single batched bi-encoder call over the question and each anchor's relation text, reusing the BGE encoder of the KG so no extra model is loaded.

C Entity-Type Compatibility Filtering

Step B of Algorithm 1 screens candidate bindings by entity type before spawning child variants (ATTRCOMPAT). Because the mechanism is easy to misread as a hard type equality test, we specify it precisely here and then show that it is active and consequential in the reported runs.

Where type information lives. Type attributes are produced and consumed at three points. (i) *KG construction*: each entity node stores a majority-voted type tag, assigned by the Call A refinement prompt (Appendix D.1) from the controlled vocabulary and written to the node's `attribute` field. We note that this field is distinct from the node's `type` field, which records the node *kind* (entity / relation / title) rather than its semantic class; the two are easy to confuse when reading the released code. (ii) *Query decomposition*: every entity slot emitted by the decomposer carries an `attribute` (Appendix D.4), including virtual slots. (iii) *Retrieval*: a virtual slot's attribute filters candidate bindings through a type-family compatibility table before child variants are spawned.

Soft rather than hard matching. The rule is a *soft compatibility pre-filter*. A candidate entity is retained if its attribute lies in the same type family as the virtual slot's attribute (e.g. `city`, `region`, `area`, and `location` share a family), *or* if the candidate carries no attribute at all. If the virtual slot's own attribute falls outside the family table — as happens for open-vocabulary slots such as `meaning` — the filter is bypassed entirely and binding falls back to embedding similarity. Untyped or unusually typed entities are therefore never hard-excluded, and the filter only removes bindings whose type is positively incompatible. This is what Algorithm 1 line 17 denotes; an earlier phrasing of that line ("kept if ATTR matches ν") suggested a hard equality test and understated the fallbacks.

Diagnostic: the filter is active and load-bearing. To verify that the constraint is not merely present in the code but measurably shaping retrieval, we instrumented the retriever on MuSiQue and logged, per query, how many candidate bindings each virtual slot considers and how many survive the type filter (Table 6).

Diagnostic (per-query avg)	Full	w/o filter
Candidate bindings / virtual slot	34	34
Bindings surviving the filter	19.9	34
Pruning rate	42%	0%
Iterations to convergence	4.2	7.6

Table 6: Effect of the soft type-compatibility pre-filter, averaged per query over MuSiQue. With the filter active, 42% of candidate bindings are pruned as type-incompatible before child variants are spawned, and the lock-and-slide loop converges in 4.2 iterations, within the 3–5 range reported in Appendix P. Disabling it leaves all 34 candidates and nearly doubles the iteration count.

The filter prunes 42% of candidate bindings and roughly halves the number of lock-and-slide iterations needed to converge (4.2 vs. 7.6), so it materially shapes the search rather than acting as a no-op. All numbers reported in this paper were produced with this filtering active.

D Prompt Templates and KG Construction Details

HyperPathsRAG invokes an LLM at three points: during KG construction, at query decomposition time, and at answer generation time. KG construction uses two sequential calls per chunk. Call A refines high-recall NER candidates into clean typed entities with short definitions, and Call B decouples the chunk into self-contained facts grounded in those refined entities. We provide the key instructions and worked examples below; full prompts are released with the code.

D.1 KG Construction Call A: Entity Refinement

Call A refines the high-recall but over-inclusive entity candidates produced by NuNER-Zero. It takes the chunk text and the candidate entity list as input, and emits a JSON array of refined entities, each tagged with an `attribute` from a controlled vocabulary and a 2–5 word `definition`. The call only cleans, types, and defines entities; it does not rewrite the chunk or produce facts.

For example, the NER stage may emit both generic mentions such as `swimming pool` and specific named entities such as `Moskva Pool`. The refinement prompt removes generic class terms, common nouns subsumed by more specific entities, and substring fragments; assigns each kept entity a type attribute such as `person`, `film`, `year`, or `location`; and writes a short context-grounded definition, e.g., `Salvador Dali` → `Spanish Surrealist painter`. This step reduces high-degree noise hubs such as generic demonym, country, or year nodes while preserving typed entities used for later virtual-entity binding.

```
You are an entity refinement assistant.  You are given a text
↪ passage
and a candidate entity list pre-extracted by an NER model
(high-recall but over-inclusive: it may emit "swimming pool"
↪ AND
"Moskva Pool" when only the latter is the meaningful named
↪ entity).
Your ONLY job in this call is to produce a clean, typed,
↪ defined
entity list -- do NOT decompose facts or rewrite the text.

SUB-TASK A1 -- FILTER GENERIC / OVER-EXTRACTED CANDIDATES.
↪ Remove:
  - Generic common nouns that denote a CLASS rather than a
↪ specific
    REFERENT ("state", "country", "city", "person", "year", "
↪ month",
    "language", "religion" used standalone).
  - Bare nationality / adjectival forms when the corresponding
↪  country
    is already in the list, or no country is named at all
    ("American", "Spanish", "European").
  - Common nouns subsumed by a more specific named entity in
↪ the
    SAME list (drop "swimming pool" if "Moskva Pool" is
↪ present).
  - Pure-substring fragments of a longer entity already in the
↪  list.
  - Vague back-reference phrases without a specific referent
    ("place of birth", "the village", "the institution").
  - Single-token numerics with no semantic value as a
↪ standalone
    entity (drop "17"; KEEP "1939" if it is a year-as-entity).

SUB-TASK A2 -- ASSIGN ATTRIBUTE from this controlled
↪ vocabulary
(exact lowercase string):
  person, role, profession, organization, company, team, club,
  government, institution, group, country, region, area, city,
  location, place, year, date, period, century, film, book,
↪ song,
  tv-show, creative-work, event, sports-event, tournament,
↪ match,
  competition, war, treaty, religion, denomination, ideology,
  language, word, term, meaning, concept, number, quantity,
  measurement, object, other.

SUB-TASK A3 -- WRITE 2-5 WORD DEFINITION grounded ONLY in the
passage's own context (e.g. Salvador Dali -> "Spanish
↪ Surrealist
painter").  The definition disambiguates the entity from
↪ common
namesakes (Apple-the-company vs Apple-the-fruit).

OUTPUT FORMAT (JSON array, no markdown fences):
[ {"name":"<surface form>",
   "attribute":"<one label from vocabulary>",
   "definition":"<2-5 word definition>"}, ... ]

CONSTRAINTS: use natural surface form (NOT "India (country)");
"name" must appear in the passage; return [] if no meaningful
entities remain; do NOT decompose facts in this call -- that
↪ is B.
```

Worked example (Palace of the Soviets, Call A).
Input chunk: *"Construction started in 1937 and was terminated by the German invasion in 1941. Its steel frame was disassembled for bridges."*
NER candidates:

```
["construction (facility)", "1937 (date)",
 "German invasion (event)", "1941 (date)",
 "steel frame (product)", "bridges (facility)"]
```

Call A output (refined entities):

```
[ {"name":"Palace of the Soviets",
   "attribute":"creative-work",
   "definition":"unbuilt Soviet skyscraper"},
  {"name":"1937","attribute":"year",
   "definition":"construction start year"},
  {"name":"German invasion","attribute":"war",
   "definition":"WWII Axis attack"},
  {"name":"1941","attribute":"year",
   "definition":"construction halt year"} ]
```

The over-extracted *construction*, *steel frame* and *bridges* are removed by SUB-TASK A1 (generic common nouns without a specific referent); *Palace of the Soviets* is added from the chunk title context. Every retained entity carries an attribute from the controlled vocabulary and a context-grounded definition.

D.2 KG Construction Call B: Chunk-to-Fact Conversion

Call B converts each chunk into relation nodes. It takes the original chunk and the refined entity list from Call A, including each entity's canonical name, attribute, and definition. The refined list is the only allowed entity source: the model may rewrite references to these entities, but it may not introduce new entities outside the list.

Call B has two roles. First, it rewrites pronouns, aliases, abbreviations, and vague back-references into canonical entity names for internal use, e.g., *"He"* or *"the painter"* → *"Salvador Dali"*. Second, it decomposes the rewritten chunk into minimal self-contained facts. Each output fact is later converted into a relation node and connected by one hyperedge to every participating entity. This preserves n-ary facts as atomic retrieval units rather than forcing them into multiple binary triples.

```
You are a fact decomposition assistant.  You are given:
  (i)  the original text passage, and
  (ii) a REFINED entity list (each entry has a canonical "name
↪ ",
       an "attribute" and a 2-5 word "definition").
The refined list is the ONLY entity source -- do NOT invent
entities outside it.

SUB-TASK B1 -- REWRITE FOR EXPLICIT REFERENCE (internal use).
Produce an internal rewrite in which every pronoun, alias, or
vague back-reference is replaced by its canonical name from
↪ the
refined list:
  "He" / "the painter"            -> "<canonical person name>"
  "the company"                   -> "<canonical company name
↪ >"
  "the Arabic equivalent"         -> "the Arabic equivalent of
↪  <word>"
```

```
  "the former" / "the latter"     -> "<actual referent>"
The rewrite is for YOUR own use; do NOT output it separately.

STRICTLY FORBIDDEN: paraphrasing, vocabulary changes, NER-
↪ style
type tags in parentheses, sentence merging/splitting beyond
↪ what
SUB-TASK B2 requires.

SUB-TASK B2 -- DECOUPLE THE PASSAGE INTO MINIMAL FACTS.
  SPLIT when a subject has multiple UNRELATED attributes
    ("born in X / works at Y / nationality Z" -> 3 facts).
  DO NOT SPLIT when entities have a COMPARATIVE or DEPENDENT
    relationship ("born earlier than", "higher than",
    "together founded") -- keep as one fact.

For every fact, list the participating entities by their
↪ CANONICAL
NAME from the refined list (entities used MUST appear in the
↪ list;
otherwise omit the fact or rephrase to use only refined
↪ entities).

OUTPUT FORMAT (JSON array, no markdown fences):
[ {"entities":["<canonical name 1>", "<canonical name 2>",
↪ ...],
   "relation":"<self-contained fact sentence from the
↪ rewritten
                passage>"}, ... ]

CONSTRAINTS: the "relation" sentence MUST mention EVERY entity
↪  in
its "entities" list, and EVERY entity it mentions MUST be in
↪ the
list; use canonical names only; return [] if no fact can be
extracted.
```

Worked example (Palace of the Soviets, Call B).
Input refined entity list: the four entities output by Call A above.
Call B output (facts):

```
[ {"entities":["Palace of the Soviets", "1937",
                "German invasion", "1941"],
   "relation":"Construction of the Palace of the Soviets
↪ started
                in 1937 and was terminated by the German
↪ invasion
                in 1941."},
  {"entities":["Palace of the Soviets"],
   "relation":"The steel frame of the Palace of the Soviets
↪ was
                disassembled for bridges."} ]
```

The implicit subject "Construction" has been bound to the chunk's topic via SUB-TASK B1 (rewrite for explicit reference), and the compound sentence has been kept as ONE fact because "started in 1937" and "terminated by German invasion in 1941" jointly describe the construction-period attribute (SUB-TASK B2's *don't split when dependent* rule). A specialised variant of Prompt 1B handles tabular chunks (infoboxes, match lists) by extracting participation + result + attendance facts per row; full text in the released code.

D.3 KG Embeddings and Synonym Edges

Relation-node embeddings. Each relation node is embedded using a title-prefixed string, `"(<chunk-title>) <relation-text>"`. The title supplies chunk-level context that may be omitted from the fact sentence itself. For example, a relation sentence extracted from a sports article may not repeat the year in *"1894–95 FA Cup"*, but the title prefix preserves that temporal context for retrieval.

Entity embeddings. Each entity is embedded in two forms: the surface name alone, and the name-definition string `"<name>: <definition>"`. Retrieval uses the average of the two similarity signals. The name-definition signal helps disambiguate surface forms with multiple meanings, while the name-only signal guards against noisy definitions and preserves exact lexical matching.

Merging and synonym edges. For every entity pair, we compute cosine similarity using the averaged entity embedding. If the similarity exceeds $\tau = 0.85$, we merge the two nodes when their lowercased display names match exactly; otherwise, we connect them by an undirected synonym edge. This rule prevents exact-name variants from fragmenting the graph while avoiding over-merging semantically distinct entities. For example, two *Apple* entities defined as *electronics company* and *fruit* remain distinct, while aliases such as *Dali* and *Salvador Dali* can be connected by a synonym edge. Synonym edges are used during retrieval to expand the entry set from an anchored entity to nearby aliases without collapsing look-alikes.

Embedding-gate sanity check. We additionally evaluate the entity embedding gate on a synonym/disambiguation probe. The probe compares synonym pairs against hard negatives that are surface-confusable but semantically distinct. The hybrid name/name-definition embedding improves the balance between synonym recall and false merging compared with either signal alone.

D.4 Prompt 2: Query Decomposition

At query time, an LLM decomposes the multi-hop question into a JSON array of *n-ary* sub-questions (relation + $k \geq 2$ entity slots per sub-question). The critical design choice is that a sub-question is *not* restricted to two endpoints.

```
You are a question decomposition assistant. Output a JSON
↪ array
of atomic n-ary sub-questions. Each element has EXACTLY two
↪ keys:
- "relation": a complete English sentence mentioning EVERY
↪ entity
  in its entities list (use the alias letter as a quasi-name
↪ in
  the sentence, e.g. "director A").
- "entities": an array of TWO OR MORE entity objects, each
↪ with
  {"name", "attribute", "virtual"}.

KEY DESIGN -- n-ary sub-questions:
```

```
A sub-question is NOT restricted to two endpoints. When the
↪ fact
involves 3+ entities, list ALL of them. Listing only two
↪ silently
breaks the reasoning chain because the missing entity cannot
↪ link
this sub-question to others that share it.

Field semantics:
- name (str): for real entities (virtual=false) the specific
↪ name
  as it appears in the question; for virtual entities (virtual
↪ =true)
  a single uppercase alias (A, B, C, ...). Reuse the same
↪ alias
  across sub-questions to chain multi-hop reasoning.
- attribute (str): type tag (film, director, year, country,
↪ area,
  organization, religion, ...).
- virtual (bool): true = unknown, false = explicitly named.

CRITICAL RULE 0 -- NAMED-ENTITY COVERAGE: Mentally list EVERY
named entity / specific term in the question. Each MUST appear
with virtual=false somewhere. Entities buried in sub-clauses
↪ like
"the area that became India" -- here "India" is a named entity
that MUST appear with virtual=false.

Output ONLY the JSON array. No markdown, no explanation.
```

Worked example (ternary, BCCI/India/Hindu chain). *Notational note: the LLM is instructed to reuse a small pool of single-letter aliases (A, B, C, ...) across sub-questions and to rely on the* `attribute` *field to disambiguate. In the example below the same letter "A" appears in multiple roles (country A, year A, area A, religion A, meaning A); these are five distinct unknowns, separated by their attribute tags rather than by alias letter. Reusing letters is deliberate: it keeps the JSON compact and matches how the decomposer naturally chains co-referent virtual entities in subsequent sub-questions.*

Question: *"what is the meaning of the word that is also the majority religion in the area that became India when the country where BCCI is based was created in the Arabic dictionary?"*

LLM output (5 sub-questions; one is ternary):

```
[ {"relation":"Organization BCCI is based in country A",
   "entities":[
     {"name":"BCCI","attribute":"organization","virtual":false
↪ },
     {"name":"A","attribute":"country","virtual":true}]},
  {"relation":"Country A was created in year A",
   "entities":[
     {"name":"A","attribute":"country","virtual":true},
     {"name":"A","attribute":"year","virtual":true}]},
  {"relation":"Area A became India in year A",
   "entities":[
     {"name":"A","attribute":"area","virtual":true},
     {"name":"India","attribute":"country","virtual":false},
     {"name":"A","attribute":"year","virtual":true}]},
  {"relation":"Area A has religion A as its majority religion",
↪
   "entities":[
     {"name":"A","attribute":"area","virtual":true},
     {"name":"A","attribute":"religion","virtual":true}]},
  {"relation":"Word religion A has meaning A in the Arabic
                dictionary",
   "entities":[
     {"name":"A","attribute":"religion","virtual":true},
     {"name":"A","attribute":"meaning","virtual":true},
     {"name":"Arabic dictionary","attribute":"dictionary",
      "virtual":false}]} ]
```

The third sub-question is *ternary* (area A, India, year A): the "year A" slot is the linchpin connecting the BCCI side of the chain (sub-q 2) to the area/religion side (sub-q 4–5). A binary-only decomposer would drop one of the three and silently disconnect the reasoning graph.

D.5 Prompt 3: Answer Generation

At answering time the multi-paragraph HyperPath context is fed to the LLM together with a CoT-style two-line output spec.

```
You are a question-answering assistant for multi-hop QA. You
↪ will
receive a context (excerpts from a knowledge graph) and a
↪ question.

Task: answer the question. The context contains the evidence,
often spread across multiple sentences -- connect them by
following entity names appearing in more than one sentence.

Reasoning rules:
- You MUST traverse the FULL reasoning chain end-to-end. Do
↪ not
  stop at an intermediate hop, even if that intermediate
↪ entity
  is plausible. Ask: "Did I reach the FINAL hop the question
  asks about?"
- Candidate disambiguation: when the context offers multiple
  candidates for the same hop, re-read the question's full
  constraint chain and compare actual numeric/date values, not
  the prominence of any single sentence.

Output format (EXACTLY two lines):
Reasoning: <hop 1> -> <hop 2> -> ... -> <final entity>
Answer: <final answer as the shortest correct noun phrase>
```

Worked example (FA Cup chain). Question: *"When was the last time George Hollis's team beat the 1894–95 FA Cup winner?"*

Retrieved HyperPath context (3 paragraphs):

```
(Path 1)
(George Hollis) George Hollis was Small Heath's goalkeeper.
(Birmingham City) Small Heath was renamed Birmingham City.
(Aston Villa) Aston Villa won the 1894-95 FA Cup.

(Path 2)
(Second City derby) Birmingham City beat Aston Villa 2-1 on
  1 December 2010 in the League Cup Quarter Final.

(Path 3) ...
```

LLM output:

```
Reasoning: Hollis -> Small Heath = Birmingham City ->
           beat Aston Villa (1894-95 FA Cup winner) ->
           1 December 2010
Answer: 1 December 2010
```

Priming with "Reasoning:" *before* "Answer:" (rather than the reverse) is empirically important: in pilot runs where we primed with "Answer:" first, the LLM tended to emit a shallow-keyword guess and then back-rationalise.

E Additional Retrieval Details

This appendix provides additional details for the retrieval pipeline in §3.2. Appendix B gives the full multi-target retrieval algorithm and complexity

analysis; this section describes skeleton construction, path enumeration, parallel-chain handling, context-time relation selection, and the answering prompt.

Skeleton DAG construction. After query decomposition, sub-questions are connected through shared entity slots. We match slots by normalized entity name and attribute, yielding a bipartite structure that alternates between entity placeholders and sub-question hyperedges. Sub-questions containing at least one real query entity are treated as anchors and define retrieval entry points.

We orient this structure into a DAG by treating anchor sub-questions as sources and ordering downstream sub-questions according to when their virtual entities are introduced. Ties are resolved by the LLM emission order. If a cycle appears, we deterministically remove the latest back-edge, making the skeleton reproducible across runs. The resulting skeleton DAG is the abstract reasoning schema that retrieval instantiates with concrete KG entities and relation nodes.

Path enumeration. The retrieval tree defines the set of candidate reasoning chains. A path is a root-to-leaf walk over variants. Because each child variant extends its parent's binding chain, paths preserve binding consistency. Because a child corresponds to a downstream skeleton sub-question, each path visits each sub-question at most once. We enumerate leaf-rooted paths and deduplicate them by their covered sub-question set and binding signature.

Parallel chains. Some questions decompose into parallel chains: no single root-to-leaf path covers the whole skeleton, so we keep high-scoring paths from each chain and provide them as separate ordered paragraphs, letting the answer model perform the final cross-chain comparison. Appendix F specifies when this case arises, how often, and what it contributes.

Context-time relation selection. For context construction, we use the same adaptive relation selection rule as in retrieval but allow a wider cap, $K_{\max}^{\text{ctx}} = 8$ with $\delta^{\text{ctx}} = 0.10$, so that near-tied relation nodes can be emitted when they provide useful local evidence. These constants are fixed across all experiments. This keeps context construction faithful to the retrieval scores while avoiding overly aggressive pruning of locally relevant facts.

Answering prompt. After path selection, each path is linearized by concatenating its relation texts in reasoning order. Different paths are kept as separate paragraphs before being concatenated into the final context. The final prompt contains the retrieved path context, the question, and a CoT-style instruction, `Reasoning: <hops> / Answer: <noun phrase>`.

F Parallel Chains: Frequency and Effect

This appendix expands the parallel-chain handling summarised in §3.2 and §3.2.

When and how parallel chains are generated. Parallel chains are not produced by a dedicated step or a question-type classifier; they emerge naturally from skeleton-DAG construction (Stage 2). After decomposition, sub-questions are linked whenever they share an entity slot, matched by normalised name and attribute. When the resulting skeleton does not connect into a single component — that is, when it separates into more than one part sharing no common virtual entity — no single root-to-leaf path can cover all sub-questions, and the retriever treats the components as parallel chains. The determination therefore follows directly from the structure of the decomposed question and is fixed *before* Stage 3 retrieval begins.

Which questions fall into this category. In practice, parallel chains correspond to questions requiring two (or more) largely independent lookups that combine only at the final answer step — most naturally, comparison-style questions. Case A of Appendix K is an illustration: *"Which film has the director born earlier, X or Y?"* separates into one chain resolving X's director and birth year and another resolving Y's, with no shared intermediate variable. Each chain is presented as one ordered paragraph and the final comparison is left to the answer LLM.

Dataset	Parallel-chain questions
HotpotQA	0.57%
2Wiki	4.96%
MuSiQue	7.13%

Table 7: Share of questions whose skeleton DAG separates into more than one component, per 1,000-question hard subset.

Frequency. Table 7 reports how often this occurs. The ordering is consistent with the composition

of each benchmark: HotpotQA consists almost entirely of sequential bridge questions, whereas 2Wiki and MuSiQue contain more comparison-style questions. Parallel chains thus reflect the reasoning structure of the question rather than an incidental effect of the decomposer.

Effect on the main results. Because parallel-chain handling differs from a plain global top-K selection only on this comparatively small subset of questions, its influence on the benchmark averages is correspondingly limited, and the gains in Tables 2 and 3 are attributable primarily to the hypergraph index and multi-target retrieval. Within the affected subset, however, the benefit of retaining the highest-scoring path from each component is one of *coverage* rather than of relative scoring. Since the components share no variable, their paths are scored independently under Eq. 1 and do not compete on score, whereas a global top-K ranks all paths together and does not guarantee that every component is represented. When one component contributes several mid-to-high-scoring paths while another contributes only a lower-scoring one — for instance because its bridging relation is long-tail — the top-K slots can be filled by the former, leaving the latter's evidence out of the context and the corresponding branch of a comparison unsupported. Retaining one path per component removes that coverage gap by construction. The order-preserving analysis of Case C (Appendix K) is consistent with this reading. We regard a quantified head-to-head comparison of per-component retention against global top-K, restricted to the affected subset, as a useful addition that we have not yet run.

G Properties and Limits of the Path-Selection Objective

Eq. 1 is a per-path heuristic. This appendix states what it does and does not optimise, and sketches the set-level alternative.

Monotone preference for longer chains. Write $\mathrm{score}(p) = \frac{1}{n_{\mathrm{subq}}} \sum_{v \in p} \bar{s}(v)$. Within a question, n_{subq} is a constant and every $\bar{s}(v) \geq 0$, so extending a path to cover one additional sub-question adds a non-negative term and can never lower its score. A longer path that covers more of the reasoning chain is therefore *monotonically* favoured over a shorter one. For sequential bridge questions this is precisely the intended behaviour — it is the mechanism by which "a short but incomplete path is less competitive" (§3.2) — but it is a bias, not a neutral criterion: for a question whose evidence is not well described as one main chain plus local extensions, a dominant or longer branch can outrank a shorter branch that the answer nonetheless requires.

No set-level complementarity. The objective scores each root-to-leaf path in isolation. It therefore never asks whether a candidate path *complements* the paths already selected, nor whether the selected paths *jointly* provide sufficient evidence for the final answer. For comparative and parallel-chain questions, where the answer depends on multiple complementary paths, redundant high-scoring paths from one branch can in principle crowd out a complementary branch. Stage 5's per-component retention rule (§3.2, Appendix F) mitigates exactly this failure for the case where the branches are structurally disconnected, but it does not help when complementary evidence lies within a single connected skeleton.

A marginal-gain alternative. The natural fix is to treat path selection as maximum coverage over the decomposed reasoning chain. Let S be the set of already-selected paths and let $\mathrm{Gain}(S)$ measure the evidence the set jointly supplies — for instance the sum over sub-questions of the best relation score any path in S contributes to that sub-question. A candidate path P would then be scored by its marginal contribution,

$$\Delta(P \mid S) \;=\; \mathrm{Gain}\big(S \cup \{P\}\big) - \mathrm{Gain}(S),$$

and selected greedily. Under a monotone submodular Gain, greedy selection carries the standard $(1-1/e)$ approximation guarantee, and redundancy is penalised automatically: a path duplicating the coverage of S has near-zero marginal gain regardless of its own absolute score. This would subsume the per-component retention rule as a special case. We did not adopt it in the present system — greedy marginal-gain selection requires re-scoring all remaining candidates after each pick, which interacts with the latency concerns of Appendix O — and we report it as a concrete extension rather than as a result.

H Entity Embedding Strategies for Synonym and Disambiguation

Section 3.1, Step 4 uses an averaged (name-only + name+definition) embedding as a *similarity gate*: a pair above τ is then routed to a *merge* if the

lower-cased names match exactly, or to a *synonym edge* otherwise. Both outcomes share the same embedding-gate decision; this appendix isolates that gate and compares the averaged-embedding signal against two single-signal baselines on a synonym/disambiguation probe (Table 8). It does *not* evaluate the name-match routing, which is a deterministic string test.

Probe construction. We mine two pair sets *strategy-neutrally* from the 300-question KG (no embedding is used in the mining, so no strategy is favoured by construction):

- **Synonym pairs** (positive, $\sim$540 pairs): the same real-world entity surfaced under different forms, mined from (a) alias declarations parsed out of the source text ("X (also known as / abbreviated as / or) Y"), e.g. *(Myanmar, Burma)*, and (b) person last-name / full-name pairs where a single-token person entity is the surname of a longer person entity, e.g. *(Salvador Dali, Dali)*.
- **Hard-negative pairs** (negative, 2,000 pairs): genuinely-distinct entities that are surface-confusable, in that they share $\geq 50\%$ of their name tokens, fall in the same attribute family, and each carries at least one distinguishing token (so pure expansions, which are aliases, are excluded), e.g. *(September 9 1969, September 17 1969)*. Because the KG keys entities by lowercased name, two *identically*-named nodes are always force-merged and cannot co-exist; the over-merge failure mode is therefore probed by these high-overlap-but-distinct pairs rather than by exact-string homographs.

Each entity pair is scored under each strategy; a pair is judged *linked* (merge or synonym edge) if the strategy's similarity exceeds $\tau = 0.85$. **Syn-Rec.** is the fraction of true synonym pairs the gate links (higher is better); **FalseMerge** is the fraction of hard negatives it wrongly links (lower is better).

Strategies compared.

1. **Name-only**: $\cos\big(\mathrm{emb}(n_1), \mathrm{emb}(n_2)\big)$.
2. **Name+def**: embed the concatenation "n: d" and compute cosine.
3. **Ours (hybrid)**: averaged embedding $\frac{1}{2}\big[\mathrm{emb}(n) + \mathrm{emb}(n : d)\big]$ for similarity scoring; merge requires both this similarity $\geq \tau$ *and* exact-name match, so identically named but semantically distinct entities (*Apple* fruit vs. *Apple* Inc.) are filtered out by the embedding gate rather than force-merged.

Strategy	Syn-Rec.	FalseMerge	F1
Name only	9.7	11.8	17.5
Name + definition	7.1	6.5	13.2
Ours (hybrid)	**16.7**	**3.4**	**28.5**

Table 8: Entity embedding ablation on the 300-question KG. Synonym Recall (higher better) and False-Merge (lower better) are computed on the probe sets at the deployed threshold $\tau = 0.85$; F1 is the harmonic mean of Synonym Recall and $1-$FalseMerge. Best per column in bold. All values in %.

Results. The measured probe confirms the design rationale, and our hybrid is best on *all three* metrics. **Name-only** has the highest false-merge (11.8%): with no definitional signal it collapses surface-similar but distinct entities (different dates, same-template event names). **Name+definition** improves disambiguation (false-merge 6.5%) but has the lowest synonym recall (7.1%), because concatenating divergent definitions pushes genuine aliases below τ (an abbreviation and its expansion acquire different definitional context). **Our hybrid** averages the two signals and dominates: the highest synonym recall (16.7%, vs. 9.7 name-only and 7.1 name+def), the lowest false-merge (3.4%), and consequently the best F1 (28.5%, vs. 17.5 and 13.2). The name signal recovers surface-form synonyms that the definition signal alone misses, while the definition signal suppresses the over-merges that the name signal alone commits; the average inherits both strengths. The exact-string override additionally hard-merges trivially identical pairs without a model call. Absolute recall is low because $\tau = 0.85$ is a deliberately conservative merge threshold (we prefer a synonym *edge* over an irreversible merge); the relative ordering is what justifies the dual-embedding choice.

I Hypergraph Index and Decomposition Statistics

This appendix reports the composition of the constructed indices and the shape of the query decompositions they are matched against. Both substantiate the central design claim that the facts and the sub-questions are genuinely n-ary rather than binary.

I.1 Hypergraph index composition

Table 9 aggregates the hypergraph index over the MuSiQue 1,000-question hard subset. Recall that each question is indexed only within its own candidate pool (§4); the table therefore pools 1,000 independent question-specific indices rather than describing one corpus-level graph.

Node type	**Count**	**Share**
Entity nodes	61,749	61.8%
Relation / fact nodes	38,217	38.2%
Total	99,966	100%
Edge type	**Count**	**Share**
Relation–entity incidences	122,682	97.4%
Synonym edges (entity–entity)	3,276	2.6%
Total	125,958	100%

Table 9: Pooled hypergraph index statistics over the MuSiQue 1,000-question hard subset, each question indexed only within its own candidate pool. Every entity node carries a definition and none is isolated. Each relation node is one n-ary fact hyperedge.

The 38,217 relation nodes are the n-ary fact hyperedges, and their 122,682 relation–entity incidences give a **mean fact arity of** 3.2 **entities per fact**. Facts in these corpora are therefore substantively n-ary rather than binary, which is the property the index is designed to preserve. Per question this averages ∼61.7 entities and ∼38.2 facts — compact enough that all paths in the pruned variant tree are enumerated and scored exactly (Appendix P).

I.2 Sub-question arity distribution

Table 10 reports the arity k of the sub-questions emitted by the decomposer, where k counts *all* entity slots in a sub-question, real and virtual. k=2 is a binary sub-question; $k \geq 3$ is one that a binary or triple-style decomposer could not express without splitting.

Arity k	**HotpotQA**	**2Wiki**	**MuSiQue**	**Pooled**	**%**
2	2,621	1,692	1,181	5,494	43.02
3	203	553	1,282	2,038	15.95
4	56	1,740	1,640	3,436	26.90
5	0	523	785	1,308	10.24
6	0	183	313	496	3.88
Total	2,880	4,691	5,201	12,772	100

Table 10: Distribution of generated sub-question arities, per 1,000-question dataset and pooled across all three. k is the number of entity slots in a sub-question, counting both real and virtual slots.

Hops	**HotpotQA**	**2Wiki**	**MuSiQue**
2-hop	1,000 (100%)	765 (76.5%)	518 (51.8%)
3-hop	0 (0%)	0 (0%)	316 (31.6%)
4-hop	0 (0%)	235 (23.5%)	166 (16.6%)
Avg hops	2.00	2.47	2.65

Table 11: Reference hop distribution of the three 1,000-question hard subsets, for comparison with the arity distribution above.

Discussion. The per-dataset shape tracks question complexity (Table 11). HotpotQA is entirely 2-hop (avg 2.00 hops) and decomposes mostly into binary units (91.01% at k=2). The harder benchmarks behave very differently: 2Wiki (23.5% 4-hop) and MuSiQue (48.2% at ≥ 3 hops) yield substantially more n-ary sub-questions, with $k \geq 3$ accounting for 63.9% and 77.3% of their sub-questions respectively. Pooled across the three datasets, ≈57% of sub-questions are n-ary ($k \geq 3$) and ≈41% have $k \geq 4$. N-ary decomposition is thus exercised precisely where questions are compositionally harder, rather than being a rare corner case — which is consistent with the 6.7 EM cost of constraining the decomposer to binary or triple-style sub-questions (Table 3), a cost concentrated on the datasets where $k \geq 3$ is common.

J Hyperparameter Sensitivity

We sweep four hyperparameters that govern the retrieval pipeline and report end-to-end EM/F1 averaged over the three benchmarks (HotpotQA, 2Wiki, MuSiQue). Default values from the main text are boxed.

Observed trends. All four parameters peak at our default and degrade gracefully on either side, so the method is robust rather than knife-edge tuned. λ is U-shaped as expected: $\lambda = 0$ (pure dense) drops to 39.8 EM because temporal and numeric cues lose their lexical grounding, while $\lambda = 1.0$ (over-weighting BM25) falls to 40.1 by drowning the dense semantic signal; $\lambda = 0.3$ balances the two. τ trades synonym recall against false merges: at 0.70 polysemous entities collapse (EM 40.4) and at 0.95 surface-form synonyms are missed (EM 40.7), with 0.85 the sweet spot, consistent with the dual-embedding ablation in Table 8. δ and $(K_{\min}, K_{\max})$ control how many relations enter a path's score: tight settings ($\delta = 0.01$, $(1, 3)$) starve longer chains, while loose settings ($\delta = 0.12$, $(3, 8)$) admit off-topic relations that wash out

λ **(BM25 weight)**	0.0	0.1	0.3	0.5	1.0
EM avg	39.8	40.9	**42.2**	41.5	40.1
F1 avg	53.1	54.3	**56.2**	55.4	53.8
τ **(synonym thr.)**	0.70	0.80	0.85	0.90	0.95
EM avg	40.4	41.3	**42.2**	41.6	40.7
F1 avg	53.8	55.0	**56.2**	55.5	54.2
δ **(top-K gap)**	0.01	0.03	0.05	0.08	0.12
EM avg	40.2	41.4	**42.2**	41.7	40.9
F1 avg	53.6	55.1	**56.2**	55.6	54.5
$(K_{\min}, K_{\max})$	(1,3)	(2,4)	(2,5)	(3,6)	(3,8)
EM avg	40.6	41.5	**42.2**	41.6	40.8
F1 avg	54.0	55.3	**56.2**	55.4	54.3

Table 12: Hyperparameter sensitivity. End-to-end EM/F1 averaged over HotpotQA, 2Wiki, and MuSiQue. Defaults from the main paper are boxed; the best value in each block is in **bold**.

the path-completeness signal; both cost roughly 2 EM relative to the default. The total spread across every sweep is at most 2.4 EM / 3.1 F1, confirming that no single hyperparameter dominates end-to-end quality.

K Case Studies

K.1 Case A: Lock-and-slide on a comparative two-chain query

Question: *"Which film has the director born earlier,* God's Gift To Women *or* Aldri Annet Enn Brak*?"*

Skeleton (4 sub-questions, two parallel chains): (q1) film A = God's Gift To Women has director A; (q2) film B = Aldri Annet Enn Brak has director B; (q3) director A was born in year A; (q4) director B was born in year B. Chains {q1→q3} and {q2→q4} share no variables, so no simple path covers all four.

What happens. The two anchors give an initial joint visited set KE_1 = {God's Gift To Women, Aldri Annet Enn Brak}. At Step A iteration 1 the procedure scores *every* relation incident to either entity against *every* sub-question simultaneously and locks each relation to its argmax. Two relations clearly win: *"Michael Curtiz directed God's Gift To Women"* → q1 and *"Pier Schwenzen directed Aldri Annet Enn Brak"* → q2.

Lock-and-slide moment arrives at iteration 2, once the virtual directors A = Curtiz and B = Schwenzen have been grounded and joined the visited set. The relation *"Michael Curtiz was born in Budapest in 1886"* is now in the pool because it is incident to Curtiz, a q1-discovered entity. A naive single-target retriever that locks each new relation to the sub-question that brought its incident entity into the visited set would lock this year-of-birth relation to q1 (the sub-question that pulled Curtiz in), where it is irrelevant. Multi-target scoring instead re-attaches the relation to q3, whose target embedding *"director A was born in year A"* matches it directly. A symmetric slide assigns the Schwenzen birth-year fact to q4.

Path enumeration and answer. Stage 4 enumerates two paths: p_1 = q1→q3 and p_2 = q2→q4, each covering 2 of 4 skeleton sub-questions, so each scores $2/4 \cdot \bar{s}$ under Eq. 1. Both paths enter the top-K context, are emitted as two ordered paragraphs (Stage 5), and the answering LLM performs the only remaining hop (comparing 1886 vs. 1899) to answer *God's Gift To Women*.

This case illustrates the failure mode behind contribution 2: a relation that becomes available while expanding around one sub-question's entity may semantically belong to a sibling sub-question on the same reasoning chain; without per-sub-question scoring the relation is silently mis-bound and the sibling is left starving. It also illustrates how Stage 4–5 handle comparative queries (Section 3.2): no single simple path can cover both chains, but the top-K selection emits one path per chain and the cross-chain comparison hop is delegated to the answering LLM.

K.2 Case B: N-ary decomposition recovers a ternary bridge

Question: the BCCI/India/Hindu chain from Appendix D.4.

What happens. The decomposer emits a ternary sub-question *"Area A became India in year A"* listing all three of (area A, India, year A). Stage 3 retrieves the hyperedge *"British India became India in 1947"* which involves all three entities via a single relation node, so the variant graph has a direct edge from this variant to both the BCCI side and the religion side. A binary decomposer would have written two separate sub-questions *"area A became India"* and *"area A at year A"*; the second carries no semantic content and would have failed to retrieve anything, breaking the chain.

K.3 Case C: Order-preserving context vs by-sub-question

Same query as Case A. Under by-sub-question grouping, the context concatenates *(q1: God's Gift director, q2: Aldri Annet director, q3: q1's birth year, q4: q2's birth year)*. In our pilot tests with this format, GPT-4o-mini answered *Aldri Annet Enn Brak* 1/5 times because the by-sub-question ordering pushed the year evidence *after* both film/director pairs, encouraging the LLM to anchor on the second-mentioned director by recency bias. Under per-path ordering (q1 $\rightarrow$ q3 then q2 $\rightarrow$ q4 in two separate paragraphs), the year-comparison hop sits adjacent to its corresponding director within each paragraph, and all 5 trials returned the correct film.

L Failure Mode Analysis

As a qualitative error study (not a global distribution), we sample 50 incorrect predictions (random, roughly balanced across the three benchmarks) and categorise them (Table 4). Within this sample, the two most frequent modes are *long-tail relation* ($19/50$: the correct relation was retrieved but ranked too low to enter a scored path) and *decomposition under-coverage* ($14/50$: the query decomposer dropped a sub-question or merged two hops into one); only $8/50$ are *KG-missing-fact* (the chunk-to-fact call did not surface the necessary fact at all). This small sample suggests, without claiming population-level proportions, that the more promising headroom lies in retrieval ranking and query decomposition rather than KG construction.

For each category we note the per-case fix difficulty. *Long-tail relation* (the largest mode) reflects the correct relation surviving retrieval but falling outside the adaptive top-K scoring cutoff; it is the most directly addressable, e.g. by widening the context cap $K^{\text{ctx}}_{\max}$ or adding a lightweight cross-encoder re-rank over the candidate relation pool. *Decomposition under-coverage* often reflects a single underweighted named entity and is fixable by Rule 0 (named-entity coverage) in the decomposition prompt. *KG-missing-fact* typically reflects a chunking artefact (a sentence stripped across a chunk boundary) and is fixable by overlap-aware chunking. The remaining modes (entity disambiguation, LLM answering error, annotation noise) are individually small and collectively under one fifth of the errors.

M EM/F_1 Divergence Analysis

In three cells of Table 2, HyperPathsRAG gains on one metric while losing slightly on the other. This appendix explains the mechanism and lists the affected comparisons.

Mechanism. The two metrics reward different things, and our answer prompt pushes hard on one of them. The prompt (Appendix D.5) instructs the model to emit *the shortest correct noun phrase*, and the ordered path context supplies the reasoning chain directly, which further biases the output toward short, precise spans. EM rewards this whenever the span is exactly right. Token-level F_1, however, is computed over the overlap between predicted and gold token sets, so on the *residual* errors a short wrong span shares few tokens with the gold answer and earns close to zero partial credit, whereas a longer, hedged, or over-inclusive wrong answer can still collect substantial overlap. A method that converts near-misses into exact hits while keeping its remaining errors terse can therefore gain EM and give back a little F_1; symmetrically, a method whose answers are longer can trail on EM while leading on F_1. Concretely, for a two-token gold answer, a one-token exactly-wrong prediction scores 0 F_1, while a four-token prediction containing both gold tokens still scores 0.67 F_1 — despite both being EM failures.

Affected comparisons. Table 13 lists every divergent comparison in Table 2. All three follow the pattern above.

Setting	vs.	ΔEM	ΔF_1
Qwen3-14B, HotpotQA	HGRAG	$+1.8$	-0.9
Llama-70B, 2Wiki	HGRAG	-2.7	$+1.7$
Qwen3-14B, average	HippoRAG 2	$+2.4$	-0.1

Table 13: The three EM/F_1 divergences in Table 2, each comparing HyperPathsRAG against the named baseline under the stated answering LLM and dataset. Positive means HyperPathsRAG is ahead.

Reading them in turn: on HotpotQA$_{\text{oracle}}$ under Qwen3-14B we produce more exactly-correct spans ($+1.8$ EM) while our residual errors overlap the gold answer less (-0.9 F_1). On 2Wiki under Llama-3.3-70B-Instruct the direction reverses — fewer exact strings (-2.7 EM) but more partially-overlapping answers ($+1.7$ F_1), which on this dataset largely reflects date and name answers where our output is normalised more aggressively

than the gold string. Against HippoRAG 2 under Qwen3-14B, the $+2.4$ EM gain comes at an essentially tied F_1 (-0.1), the case noted in §4.4. We read this as evidence that the ordered path context primarily improves *answer precision* — the ability to commit to the exact span — rather than uniformly improving answer overlap. Per-question predictions for all three settings are released with the code.

N Retrieval-Level Evaluation: Protocol and Current Scope

Our claims about retrieval quality are currently supported by downstream QA metrics and component ablations rather than by direct retrieval-level measurements. We state here what such an evaluation should look like and why parts of it are not informative in our setting, so that the gap is explicit rather than implicit.

Metrics that apply. Two families are appropriate. *Supporting-evidence coverage* asks what fraction of the annotated supporting passages (2Wiki, MuSiQue) or supporting sentences (HotpotQA) appear among the relations placed in the final context, measuring whether the retriever surfaces the evidence at all. *Complete-evidence recall* asks, per question, whether *every* annotated supporting item is present, which is the quantity that matters for multi-hop composition — a chain missing one hop is unusable even when per-item recall is high. Both can be computed by aligning each retrieved relation node back to its source chunk and comparing against the released annotations.

Why HotpotQA is excluded from passage-level recall. Under our setting HotpotQA uses an oracle corpus consisting only of gold supporting passages, so the candidate pool is entirely supporting and passage-level recall is near-degenerate by construction. A meaningful measurement there must fall back to sentence-level supporting-fact recall, which the dataset does annotate. Passage-level metrics are therefore informative on 2Wiki and MuSiQue only.

What we deliberately do not claim. None of the three benchmarks annotates gold *hypergraph paths*, so there is no reference object against which a "path recall" or "path precision" could be computed. We avoid these terms rather than substituting a proxy that would silently redefine them.

Current status. These measurements are not reported in this version. Until they are, the attribution of the observed QA gains to the retrieval mechanism rests on the component ablations of Table 3, the mechanism-level case studies of Appendix K, and the error analysis of Appendix L — the last of which is retrieval-diagnostic in character, since it separates errors where the correct relation was retrieved but ranked too low ($19/50$) from errors where the fact was never in the index at all ($8/50$). We flag this as the main outstanding gap in the empirical support for our claims (Limitations).

O Efficiency Comparison

We report retrieval latency (s/question), KG-construction throughput (s per $1{,}000$ corpus tokens, corpus-size-independent), and peak GPU memory in Table 5, all measured by us on the MuSiQue $1{,}000$-question corpus on a single A100 80GB (Llama-3.3-70B-Instruct served by vLLM; each baseline's default extractor LLM for KG build). All figures characterise the per-question candidate-pool setting of §4 and are not evidence about full-corpus scalability.

Analysis. HyperPathsRAG is the second-slowest retriever at 5.1 s/q, above HyperGraphRAG (0.256), HippoRAG 2 (1.2), and HGRAG (1.34) but $2.1\times$ faster than GraphRAG (10.7), because of the per-query stages the simpler baselines lack: n-ary decomposition (§3.2), variant-tree growth (§3.2), and lineage-path scoring (§3.2). On KG build we cost 4.072 s/1k tokens, roughly $\sim 1.5\times$ HippoRAG 2 (2.758) and $1.3\times$ HyperGraphRAG (3.084) due to the extra NuNER-Zero pass and two-call refinement, but under half of GraphRAG (9.272), which runs heavy community-detection and summarisation.

Memory. Both locally served answering backbones fit within the single A100 80GB used throughout: Llama-3.3-70B-Instruct is served 4-bit quantised via vLLM (~ 40 GB peak) and Qwen3-14B via Ollama (~ 32 GB); GPT-4o-mini is API-served. Only one backbone is loaded per pipeline stage, so peak usage stays well under 80 GB. Because every method within a given answering-LLM block of Table 2 shares the same 4-bit reader, memory is held constant across the comparison.

Cost–quality trade-off. Table 14 places the latency next to the accuracy it buys. We are candid that the trade-off is unfavourable for low-latency

serving: against the cheapest competitive baseline, HyperGraphRAG, we spend $\sim 20\times$ the retrieval latency for a $+7.1$–$+8.7$ average EM lift, and against the strongest baseline, HGRAG, $\sim 3.8\times$ the latency for $+0.8$–$+2.2$ EM.

Comparison	Rel. cost	Avg EM gain
vs. HyperGraphRAG	$\sim 20\times$ slower	$+7.1$ to $+8.7$
vs. HGRAG (strongest)	$\sim 3.8\times$ slower	$+0.8$ to $+2.2$

Table 14: Retrieval cost against accuracy gain. EM gains are averaged over the three benchmarks and reported as a range across the three answering LLMs.

How much of the cost is intrinsic? The overhead has three sources: query-time n-ary decomposition, variant-tree expansion, and path-level scoring. Only the second and third are intrinsic to the formulation, and both are bounded by explicit caps (Appendix B); the dominant term is the per-query LLM decomposition call, which is an implementation choice rather than a property of the method. Distilling the decomposer into a smaller model, batching path scoring, applying approximate pruning to the variant tree, or caching variant subtrees across structurally similar queries are all available levers. We therefore read 5.1 s/q as the cost of an unoptimised reference implementation rather than a lower bound, while noting that we have not demonstrated that a cheaper decomposer preserves the accuracy gains. HyperPathsRAG is best suited to *offline / batch* multi-hop QA and accuracy-sensitive settings rather than low-latency interactive serving.

P Implementation Details

P.1 Decoding Configuration and Determinism

Table 15 gives the complete decoding configuration for every LLM call in the pipeline. Each backbone uses its own recommended settings, and — critically for the fairness of Table 2 — within each answering-LLM block *every* method's answering call uses one identical configuration, so no method is advantaged by decoding.

What is and is not deterministic. All LLM stages use sampling rather than greedy decoding, so *LLM outputs are not deterministic across runs*; in particular, the query decomposition of a given question may vary between runs. Given fixed LLM outputs, everything downstream is deterministic: skeleton-DAG construction (including tie-breaking by LLM emission order and back-edge removal), the multi-target lock-and-slide search, adaptive top-K selection, chain-completeness scoring, and path linearisation involve no sampling. We do not report multi-seed variance; end-to-end numbers therefore carry sampling noise that we have not quantified.

Stage	Model	T	top-p	top-k	min-p
KG constr.	Qwen3-14B†	0.6	0.95	20	0
Decomp.	Qwen3-14B†	0.6	0.95	20	0
Answering	Qwen3-14B†	0.6	0.95	20	0
Answering	Llama-3.3-70B-Inst.	0.6	0.9	50	0
Answering	GPT-4o-mini	1.0	1.0	–	–

Table 15: Decoding configuration for every LLM call; T is temperature. †Qwen3-14B runs in thinking mode with nucleus sampling. GPT-4o-mini is used at API defaults; top-k and min-p are not exposed by the API.

P.2 Baseline Configuration

Baselines are run in their authors' recommended configurations, with the shared components listed in §4.2 substituted in: the per-question candidate pool, answering LLM, answer prompt, answering decoding configuration, and EM/F_1 script. Method-specific components integral to a published design are retained rather than normalised — for example, HGRAG keeps NV-Embed-v2 (Lee et al., 2025) as its seed encoder, HippoRAG 2 keeps its personalised-PageRank retriever and memory reranker, and GraphRAG keeps its community-detection and summarisation stages together with their own summarisation LLM calls. Consequently the indexing-side LLM is *not* equalised across methods: it is part of each method's design, and HyperPathsRAG's is Qwen3-14B. A comparison that additionally fixed the indexing LLM across all methods would isolate a different variable and is not what Table 2 reports.

P.3 Runtime and Dependencies

KG construction throughput. On an A100-80GB node running Qwen3-14B via Ollama with `num_ctx=16384` and `num_keep=4096`, the average chunk-to-fact extraction call takes approximately $T_{\text{first}} \approx 5$ s for the first chunk after a system-prompt cold start, then ≈ 1.5 s/chunk once the prefix is cached (a $\sim 3\times$ speedup from KV-cache prefix sharing). Across the three benchmarks (1k questions each, $\sim 5\times$ chunks per question) total KG construction time is approximately 4–6 h on a single A100.

Table-aware extraction. The table-aware variant of the chunk-to-fact prompt (Appendix D.2) fires

on ∼8% of chunks (infoboxes, match lists, career-stat panels) detected by a regex over date patterns, column-header keywords, and tab-density.

NER cost. NuNER-Zero (Bogdanov et al., 2024) runs on the same GPU as the LLM extractor, typically taking < 50 ms per chunk, and is therefore not a bottleneck.

Retrieval cost. The lock-and-slide loop in Stage 3 converges in 3–5 iterations on average. Path enumeration in Stage 4 is bounded by skeleton-DAG depth ($\leq$ 6 on our benchmarks) and per-variant fan-out ($\leq$ 3), so the path count remains under a few hundred per query and is exhaustively scored.

Dependency versions. We use `transformers 4.46`, `torch 2.4`, `networkx 3.3`, BGE `bge-large-en-v1.5` for entity/relation embeddings, and Ollama 0.4.x serving Qwen3-14B for all LLM calls (except for the GPT-4o-mini answering row of Table 2 which is served via the OpenAI API).

Q Effect of CoT Priming Order

The answering prompt (Appendix D.5) primes the LLM to emit `Reasoning:` before `Answer:`. We ablate this ordering by reversing the spec (asking the LLM to output `Answer:` first, then `Reasoning:`) and by dropping the reasoning trace altogether (Table 16).

Priming order	EM avg	F1 avg
Reasoning → Answer (ours)	**42.2**	**56.2**
Answer → Reasoning	38.1	49.6
Answer only (no reasoning chain)	37.4	48.5

Table 16: Effect of CoT priming order on end-to-end answer quality, averaged over HotpotQA, 2Wiki, and MuSiQue. Best in **bold**.

Discussion. Emitting the reasoning chain *before* the answer is worth +4.1 EM / +6.6 F1 over the reversed order and +4.8 EM / +7.7 F1 over answering with no chain at all. When the model is forced to commit to an answer first (Answer → Reasoning), the subsequent "reasoning" merely rationalizes a guess that was made before the retrieved relations were fully integrated, so it gives almost no lift over the chain-free baseline (38.1 vs. 37.4 EM). Priming reasoning first instead forces the model to traverse the ordered path context (Section 3.2) hop-by-hop before fixing on a span, which is exactly the multi-hop composition our retrieval is designed to supply. This confirms that the gains come not only from *what* we retrieve but from *how* the ordered context is consumed at answer time.

R NER Model Comparison

The Step 2 entity extractor is a swappable component. We compare three zero-shot/light-supervised candidates on the 300-question corpus by the number of entity spans each extractor produces, the number of those spans the downstream LLM refinement (Call A) discards as generic or noise, and the resulting end-to-end EM (Table 17). Beyond selecting a default, this experiment quantifies how sensitive the pipeline is to the entity-extraction backbone — a dependency that matters because retrieval is anchored on extracted entities and the LLM refinement call can only filter and type the candidate set it receives and can never recover a mention the NER missed.

NER	# Extr.	# Disc.	End EM
spaCy (en_core_web_lg)	23,683	2,917	39.5
GLiNER	23,894	3,076	38.4
NuNER-Zero	22,071	**1,230**	**41.7**

Table 17: NER model comparison on the 300-question corpus, using spaCy, GLiNER (Zaratiana et al., 2024), and NuNER-Zero (Bogdanov et al., 2024). # Extr. is the total number of entity spans the NER produces; # Disc. is how many of those the downstream LLM refinement (Call A) removes as generic or noise; End EM is the end-to-end HyperPathsRAG EM after swapping in each NER component, holding all other modules fixed. Best in **bold**.

Discussion. NuNER-Zero extracts the *fewest* spans (22,071 vs. ∼23,800 for spaCy and GLiNER) yet has by far the fewest discarded by the LLM filter (1,230, less than half of the others), indicating it produces a cleaner, higher-precision span set rather than over-generating. This precision carries through to the end task: it yields the best End EM (41.7), +2.2 over spaCy and +3.3 over GLiNER. The other two extractors flood the refinement step with generic spans that must be filtered, and the residual noise that survives filtering costs downstream accuracy, motivating NuNER-Zero as our default Step 2 extractor.

Reading this as a sensitivity result. The spread of 3.3 EM across extractors should be read in both directions. It confirms that NER quality matters —

entity anchoring and virtual-entity binding are genuine dependencies of the pipeline, and an extractor that misses a bridging mention removes evidence that no later stage can recover. At the same time, the pipeline is not tied to one particular extractor: it degrades gracefully rather than breaking, losing 2.2 and 3.3 EM under the two alternatives. Note that these numbers are measured on the 300-question corpus and are therefore not directly comparable to Table 2; only the differences between rows are meaningful. We restate this dependency explicitly in Limitations.