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

Anonymous ACL submission

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://anonymous.4open.science/r/HyperPathsRAG-AE05/`.

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; **?**). 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., 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., 2026a; 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 (**??**Gong et al., 2025; 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. Candidate paths are scored by their coverage of the decomposed reasoning chain, and the selected

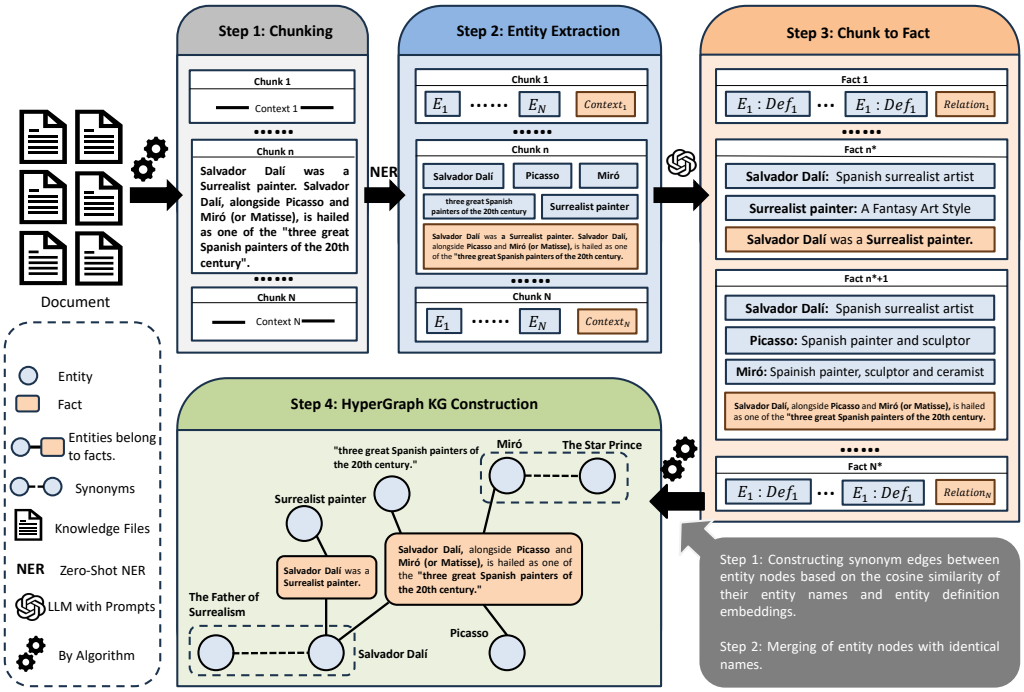

Figure 1: Hypergraph KG construction. **Step 1 (Chunking)** splits each document into non-overlapping chunks. **Step 2 (Entity Extraction)** runs zero-shot NER followed by an LLM refinement call that filters generic candidates, assigns type attributes, and writes short definitions for kept entities. **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.

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., 2020; **?**). 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., 2025; Fan et al., 2025; Zhu et al., 2025; **?**; Huang et al., 2025; Luo et al., 2026b; **?**; Guo et al., 2025). 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., 2026a; 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., 2022; 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 (**??**), while later work refines sub-question schemas or iterates retrieval with LLM feedback (Gong et al., 2025; Wang et al., 2026b; Sorokin et al., 2025; Hui et al., 2025; 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 B.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 B.2 and Appendix B.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 B.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

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

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 hyper-

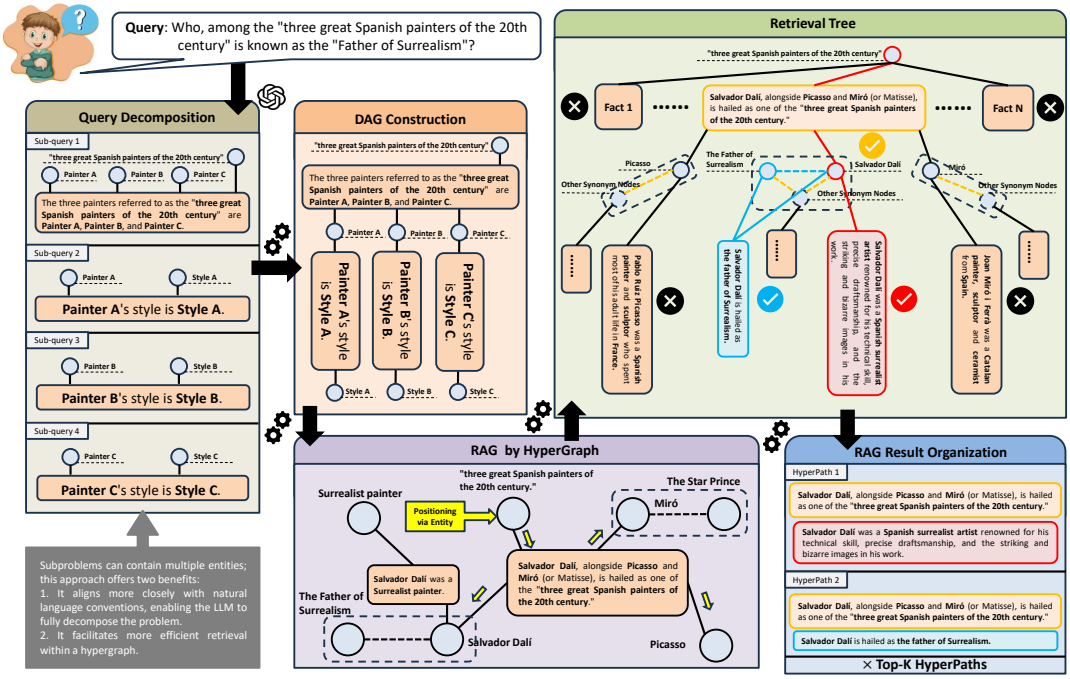

Figure 2: The retrieval pipeline. An LLM decomposes the multi-hop query into a list of n-ary sub-questions (top-left, each carrying one or more entity placeholders). Sub-questions sharing entities are joined into a *skeleton DAG*; real-entity sub-questions become anchor nodes. The hypergraph KG is then 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).

graph. The decomposition prompt, JSON schema, and examples are given in Appendix B.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 C.

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. Full pseudocode, hybrid relation scoring, adaptive cutoffs, and complexity are given in Appendix A.

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

Baseline	Retrieval strategy
No retrieval and chunk retrieval	
NaiveGeneration	Answers directly from the LLM without retrieval.
StandardRAG (Lewis et al., 2020)	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., 2025)	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., 2026a)	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.

$p = (v_1, \ldots, v_m)$ 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 C.

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 J.

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. HotpotQA uses only the gold supporting passages, forming an *oracle-document* corpus ("$\text{HotpotQA}_{\text{oracle}}$"), while 2WikiMultiHopQA (4+6) and MuSiQue (4+16) use the standard distractor pools. The gold answer is never visible at retrieval.

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 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 prefiltering followed by dense reranking, and relation

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).

scoring combines dense similarity with a normalized BM25 boost. Entity and relation embeddings are computed with BGE-large-en-v1.5. 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 A.

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, runtime settings, dependency versions, and timing breakdowns are provided in Appendix I.

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.

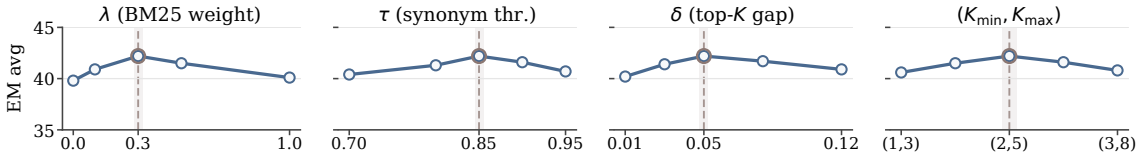

Figure 3: Hyperparameter sensitivity. Each panel varies one retrieval hyperparameter while keeping the others fixed. The vertical dashed line marks the default used in the main experiments. EM remains within a narrow range across all sweeps, suggesting that HyperPathsRAG is not brittle to a narrow hyperparameter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

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.

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.

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.

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

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.

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.

presented in reasoning order. These drops should not be interpreted additively, since the components 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 J). 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 E.

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 G 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. 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 H gives the full cost–quality discussion, and Appendix I gives implementation-level timing details.

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 three MHQA benchmarks and three answering LLMs, HyperPathsRAG achieves the best average EM, 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 conducted in the *candidate-passage* setting rather than full-corpus open-domain retrieval. HotpotQA uses an oracle-document corpus, while 2WikiMultiHopQA and MuSiQue use released distractor pools. This setting follows recent Graph-RAG evaluations and enables controlled comparison, but it does not measure the cost or recall of building and searching a corpus-level hypergraph index. Extending HyperPathsRAG to full-corpus retrieval is therefore an important direction for future work.

References

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 1 others. 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, Carl Yang, and 1 others. 2025. Beyond chunks and graphs: Retrieval-augmented generation through triplet-driven thinking. *arXiv preprint arXiv:2508.02435*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv 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 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. *arXiv preprint arXiv:2503.10150*.

Yulong Hui, Chao Chen, Zhihang Fu, Yihao Liu, Jieping Ye, and Huanchen Zhang. 2025. Interact-rag: Reason and interact with the corpus, beyond black-box retrieval. *arXiv preprint arXiv:2510.27566*.

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, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Haoran Luo, Guanting Chen, Yandan Zheng, Xiaobao Wu, Yikai Guo, Qika Lin, Yu Feng, Zemin Kuang, Meina Song, Yifan Zhu, and 1 others. 2026a. Hypergraphrag: Retrieval-augmented generation via hypergraph-structured knowledge representation. *Advances in Neural Information Processing Systems*, 38:152206–152234.

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

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.

Artyom Sorokin, Nazar Buzun, Alexander Anokhin, Oleg Inozemcev, Egor Vedernikov, Petr Anokhin, Mikhail Burtsev, Trushkov Alexey, Yin Wenshuai, and Evgeny Burnaev. 2025. Q-rag: Long context multi-step retrieval via value-based embedder training. *arXiv preprint arXiv:2511.07328*.

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.

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.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv 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*.

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.

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 $\mathcal{T}$

$\mathcal{T} \leftarrow \varnothing$; queue $\mathcal{Q} \leftarrow [\,]$
for each anchor sub-question $q_a \in \mathcal{A}$ **do**
  $E \leftarrow$ BM25-then-dense lookup of q_a's real entities
  $v_a \leftarrow (qid{=}q_a,\ bind{=}\varnothing,\ anchor{=}E \cup \textsc{SynClosure}(E))$
  add v_a to $\mathcal{T}$; enqueue v_a
end for
while $\mathcal{Q} \neq [\,]$ **and** $|\mathcal{T}| < \textsc{MaxVar}$ **do**
  pop group $\mathcal{G}$ of variants sharing the same *anchor*
  // Step A: ancestor contention with multi-lock
  $R \leftarrow \bigcup_{e \in anchor(\mathcal{G})} \textsc{Rel}(e) \cup \textsc{ComposedRetrieve}(Q, \mathcal{G})$
  $\mathcal{C} \leftarrow \bigcup_{v \in \mathcal{G}} \textsc{Ancestors}(v)$ ▷ contestants
  for each contestant $a \in \mathcal{C}$ **do**
    score every $r \in R$ against q_a by dence-sparse score
    $lock(a) \mathrel{+}= \textsc{AdaptiveTopK}(R; K_{\min}, K_{\max}, \delta_{\text{rel}}, \tau_{\text{rel}})$
    ▷ no single-winner: r may be locked by several a
  end for
  // Step B: virtual binding → spawn children
  for each $v \in \mathcal{G}$ with newly locked relations **do**
    for each unbound virtual ν in v **do**
      $C \leftarrow$ non-anchor endpoints of v's locked relations, kept if $\textsc{Attr}$ matches ν
      $B \leftarrow \textsc{AdaptiveTopK}(C; \cdot, N_{\max}, \delta_{\text{bind}}, \tau_{\text{bind}})$ by avg. name/name+def sim
      for each $e \in B$ **do**
        $q' \leftarrow$ next skeleton sub-question sharing ν
        $v' \leftarrow (q',\ bind(v) \cup \{\nu{\to}e\},\ \{e\} \cup \textsc{SynClosure}(e))$
        if $(q', bind(v')) \notin \mathcal{T}$ **then** add v' to $\mathcal{T}$; enqueue v'
        end if
      end for
    end for
  end for
end while
return $\mathcal{T}$

A 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. $O(dL)$ scoring operations; binding spawns at most $N_{\max}$ children per virtual. The variant count is bounded by the explicit cap $\textsc{MaxVar}$, so the whole loop is $O(\textsc{MaxVar} \cdot dL)$ 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. I). $\textsc{ComposedRetrieve}$ 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.

B 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.

B.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.

B.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.

B.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.

B.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.

B.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.

C Additional Retrieval Details

This appendix provides additional details for the retrieval pipeline in §3.2. Appendix A 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. For example, comparative questions may require retrieving two entities independently before the answer model compares them. In such cases, no single root-to-leaf path necessarily covers the whole skeleton. We therefore keep high-scoring paths from each chain and provide them as separate ordered paragraphs, allowing the answer model to perform the final cross-chain comparison from the retrieved evidence.

Context-time relation selection. For context construction, we use the same adaptive relation selection rule as in retrieval but allow a wider cap, $K^{\text{ctx}}_{\max} = 8$ 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>`.

D 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 6). 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**: $\cos\big(\text{emb}(n_1), \text{emb}(n_2)\big)$.
2. **Name+def**: embed the concatenation "n: d" and compute cosine.
3. **Ours (hybrid)**: averaged embedding $\frac{1}{2}\big[\text{emb}(n) + \text{emb}(n : d)\big]$ 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 6: 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.

E 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 6. 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 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.

λ (BM25 weight)	0.0	0.1	$\boxed{0.3}$	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	$\boxed{0.85}$	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	$\boxed{0.05}$	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)	$\boxed{(2,5)}$	(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 7: 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**.

F Case Studies

F.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"* $\rightarrow$ q1 and *"Pier Schwenzen directed Aldri Annet Enn Brak"* $\rightarrow$ 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$\rightarrow$q3 and p_2 = q2$\rightarrow$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.

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

Question: the BCCI/India/Hindu chain from Appendix B.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.

F.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.

G 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.

H Efficiency Comparison

We report retrieval latency (s/question) and KG-construction throughput (s per 1,000 corpus tokens, corpus-size-independent) against three baselines 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).

Analysis. HyperPathsRAG is the second-slowest retriever at 5.1 s/q, above HyperGraphRAG (0.256) and HippoRAG 2 (1.2) but 2× 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). This 4–20× slowdown over the lighter baselines buys the +0.8–+2.2 EM gain of Table 2. On KG build we cost 4.072 s/1k tokens, roughly ∼1.5× HippoRAG 2 (2.758) and 1.3× 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.

Cost–quality trade-off. We are candid that the per-query gain is modest relative to the cost: against the cheapest competitive baseline HyperGraphRAG, we spend ∼20× the retrieval latency for a +0.8–+2.2 EM lift, dominated by the per-query LLM decomposition call. HyperPathsRAG is therefore best suited to *offline / batch* multi-hop QA and accuracy-sensitive settings rather than low-latency interactive serving; reducing the decomposition cost (e.g. distilling it into a smaller model or caching variant subtrees across structurally similar queries) is the most impactful efficiency lever and is left to future work.

I Implementation Details

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 ∼3× speedup from KV-cache prefix sharing). Across the three benchmarks (1k questions each, ∼5× 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 B.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 (≤ 6 on our benchmarks) and per-variant fan-out (≤ 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).

J Effect of CoT Priming Order

The answering prompt (Appendix B.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:`).

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 8: 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 re-

lations 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.

K 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.

NER	# Extracted	# Discarded	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 9: NER model comparison on the 300-question corpus; citations as in Appendices B.1–I. # Extracted is the total number of entity spans the NER produces; # Discarded 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. $\sim$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.