DOES AGENTIC RAG OVER KNOWLEDGE GRAPHS HELP SCIENTIFIC QA? ONLY WHERE NO SINGLE QUERY REACHES THE EVIDENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Retrieval-augmented generation grounds a language model in a corpus, but one retrieval call answers one kind of question. Questions that bridge two papers, aggregate a quantity reported across several, or combine a numeric detail with an entity relationship need evidence a single dense query does not return. Graph-based retrieval already offers the operations such questions require: dense vector search, entity-centred graph traversal, and relationship-centred synthesis. A deployed system, however, fixes one of them per query. We present MODERAG, in which a Planner assigns each sub-task the retrieval mode its part requires and declares the dependencies between sub-tasks, an Executor runs ready sub-tasks in parallel, and a Reporter answers from an evidence chain that records the chunks, graph paths and entities behind each step. We release `LHCpubs-100QA`, 100 multi-paper questions over LHC papers with annotated evidence, and evaluate alongside `AI4EIC2023`, 50 mostly single-paper questions. ModeRAG improves answer correctness by 7.7 points over the same pipeline restricted to vector search, with the largest gains on questions that bridge or aggregate across papers; the difference is unresolved on the single-paper benchmark. Dense retrieval remains the backbone, and decomposition costs 24,268 tokens per question.

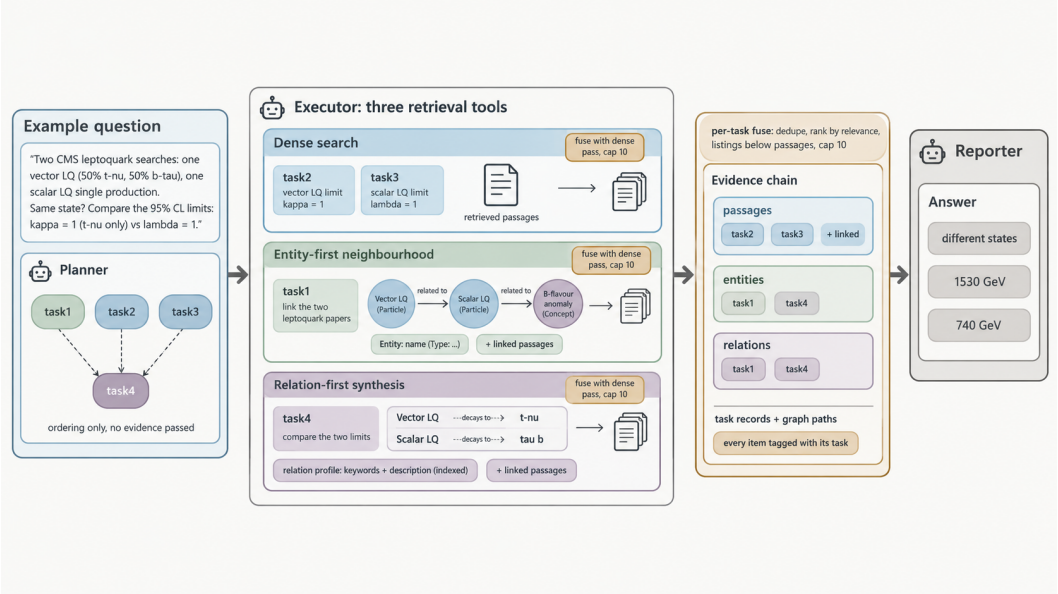


Figure 1: ModeRAG overview. The Planner specifies retrieval tasks before execution, the Executor schedules ready tasks, and the Reporter answers from provenance-bearing evidence; dependencies control scheduling but do not pass intermediate retrieval results to later sub-queries.

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

1 INTRODUCTION

Retrieval-Augmented Generation (RAG) grounds language-model answers in an external corpus (Lewis et al., 2020; Ram et al., 2023). In a standard system, one dense query retrieves a small set of text chunks for generation. This is effective when a relevant passage contains the answer, but complex questions may require several facts, entities, or relationships that cannot all fit in the generator's context or be synthesized from a single retrieval result. The bottleneck is evidence coverage and integration, even when the relevant papers are present in the corpus-level candidate pool.

Scientific questions often combine different evidence demands. A comparison of measurements across experiments may require exact values from separate papers, entity-centered links between detector components and observables, and a synthesis of the resulting relationships. Dense retrieval, graph traversal, and relationship-level synthesis offer complementary access patterns; applying one uniformly can leave parts of a question unsupported.

GraphRAG systems organize corpora into entities and relationships (Edge et al., 2024; Guo et al., 2024; Baek et al., 2023; He et al., 2024). LightRAG supports naive, local, global, and native hybrid retrieval, as well as a mix mode that adds vector-chunk retrieval to the graph path; its hybrid mode combines local and global graph retrieval in one query (Guo et al., 2024). Multi-agent systems decompose complex requests into tasks (Wu et al., 2024; LangChain, 2024), and interleaved retrieval and reasoning is also well established (Yao et al., 2023; Trivedi et al., 2023). ModeRAG investigates a different design question: whether planning a set of retrieval-mode-specific sub-queries, scheduling their dependencies, and tracking their evidence improves answer quality on scientific literature. The external comparison is therefore against LightRAG's native hybrid retrieval, which combines local and global graph access in a single call (Section 4.1); we do not treat single-mode retrieval as representative of existing GraphRAG systems.

We present MODERAG, a Planner–Executor–Reporter framework with three retrieval tools: dense vector search (naive), entity-centered graph traversal (local), and relationship-centered synthesis (global). The Planner generates dependency-annotated tasks before execution; the Executor runs ready tasks concurrently; and the Reporter answers from a provenance-bearing evidence chain. The dependencies govern scheduling only: sub-queries are fixed in advance, and intermediate results do not condition later queries. The prompted acyclic structure is not validated by the implementation (Appendix A).

Our core contributions are as follows:

1. We present MODERAG, a provenance-aware Planner–Executor–Reporter framework that assigns pre-generated sub-queries to dense, entity-centered, or relationship-centered retrieval and schedules tasks by their declared dependencies. Its dependencies govern scheduling only; queries are fixed before execution.
2. We release the `LHCpublications` corpus: INSPIRE metadata for 3,589 ATLAS/CMS records, 2,771 papers converted to full markdown text with figures, 399 documented extraction failures, and the pipeline that rebuilds it.
3. On a 96-paper indexed subset of that corpus we release `LHCpubs-100QA`, 100 questions annotated with four cross-paper evidence genotypes, 25 per genotype, together with the retrievability gate, the generation pools, and the scripts and prompts that construct it. We evaluate alongside 50 mostly single-paper questions from `AI4EIC2023`.
4. We provide a diagnostic evaluation of when this configured system helps. On the released LHC set, All Modes improves correctness by $+0.077$ over the prompt-matched Naive Only system, while the difference on AI4EIC2023 is unresolved at the current sample size. These results are specific to the evaluated corpora and to the internal ablation protocol.

2 BACKGROUND AND RELATED WORK

Retrieval-augmented generation. Retrieval-Augmented Generation (RAG) conditions generation on documents retrieved from an external corpus (Lewis et al., 2020; Ram et al., 2023): an indexer maps the corpus into a searchable structure, a retriever returns its query-relevant content, and a generator answers from it. In the standard dense pipeline the indexer embeds chunks into a

vector space and the retriever performs approximate nearest neighbour search over them, usually in a two-stage form that recalls top-k_1 candidates and re-ranks them to a final top-k_2. This works when the answer lies in one chunk, but questions that require structured reasoning over entities and relationships spanning several documents are not served by the flat representation; self-correcting retrieval (Yan et al., 2024) addresses retrieval errors rather than that representation.

Knowledge-graph-augmented retrieval. Graph-based RAG addresses that limitation by organizing the corpus into a knowledge graph whose nodes are entities and whose edges are typed relationships (Edge et al., 2024; Guo et al., 2024). Following LightRAG (Guo et al., 2024), construction prompts an LLM to recognize entities and relationships within each chunk, attaches descriptive key–value profiles for efficient retrieval, and merges co-referential entities. The resulting graph enables two capabilities absent in flat RAG: extracting global information from multi-hop subgraphs, and retrieving entities and relationships directly, which is more precise than embedding matching (Gao et al., 2022; 2023) and more efficient than brute-force community traversal (Edge et al., 2024). It complements rather than replaces the text index, since extraction can omit or merge domain-specific concepts while source passages preserve the exact wording and numerical context needed to verify a scientific claim. Systems consequently differ in what they retrieve: Microsoft GraphRAG builds corpus-level communities and summaries for query-focused synthesis, HippoRAG uses associative graph retrieval for multi-hop evidence (Gutiérrez et al., 2024), and LightRAG exposes local and global graph access in a native hybrid call; others differ in how the graph is built or traversed (Baek et al., 2023; He et al., 2024).

Multi-mode retrieval over knowledge graphs. A key design decision is *how* to search the graph. LightRAG (Guo et al., 2024) supports naive vector retrieval, local entity-centric traversal, global relationship-centric synthesis, a hybrid mode that combines local and global retrieval for one query, and a mix mode that adds vector-chunk retrieval to the graph path. These operations address different evidence needs: factual and numeric questions are served from individual chunks, entity-centric questions need relationships around specific entities, and comparative questions need synthesis across documents and relations. The open question is therefore not whether to search a graph but how to allocate several retrieval operations within one question; in the systems above the allocation is fixed per query, either by committing to a single operation or by fusing them in one call. Agentic Hybrid RAG (Jiang et al., 2026) runs an agentic pipeline over hybrid retrieval for evidence-grounded analysis of a single scientific corpus, which brings the same question into an agentic setting.

Multi-agent pipelines. Multi-agent systems decompose complex tasks into sub-tasks distributed across specialized agents. Frameworks such as AutoGen (Wu et al., 2024) and LangGraph (LangChain, 2024) provide structured control flow, and the Plan–Execute–Report pattern is common: a Planner generates a plan, Executors carry out the tasks, often in parallel, and a Reporter synthesizes the output. Interleaving reasoning with retrieval is also standard (Yao et al., 2023; Trivedi et al., 2023), and Singh et al. (2025) survey the design space; IRCoT (Trivedi et al., 2023) lets later retrieval depend on earlier evidence. These frameworks orchestrate heterogeneous capabilities well, but they are usually agnostic to the retrieval substrate, so each agent decides how to query the knowledge base, which makes grounding and evidence tracking difficult.

Evidence chains and explainability. Many RAG systems are opaque: the user receives an answer but cannot trace how it was derived from the retrieved evidence. Work on explaining and evaluating RAG pipelines (Sudhi et al., 2024; Salemi & Zamani, 2024; Es et al., 2024) typically stays within the scope of a single retrieval call, so provenance has to be reconstructed after the fact once retrieval is orchestrated across several calls.

Positioning. The systems above differ in how they build and traverse a graph and in which retrieval operations they expose. Our question is how a small set of such operations should be allocated across the sub-queries of one question: MODERAG treats them as tools, plans their assignment and dependencies before execution, and keeps every answer attributable to the passages, entities and paths that produced it. Table 1 places it against the closest lines of work; the external comparison is against a single LightRAG hybrid call, described in Section 4.1.

Method	Dense	Graph	Multi-call	Subtask dependency	Evidence chain
Vanilla RAG (Lewis et al., 2020)	✓	–	–	–	–
Microsoft GraphRAG (Edge et al., 2024)	–	✓	✓	–	–
LightRAG (Guo et al., 2024)	✓	✓	–	–	–
HippoRAG (Gutiérrez et al., 2024)	✓	✓	–	–	–
IRCoT (Trivedi et al., 2023)	✓	–	✓	–	–
Agentic Hybrid RAG (Jiang et al., 2026)	✓	–	✓	–	✓
ModeRAG (ours)	✓	✓	✓	✓	✓

Table 1: Positioning. *Dense*, *Graph*: the operation is exposed as a retrieval tool. *Multi-call*: one query can issue more than one retrieval or planning call. *Subtask dependency*: retrieval sub-tasks are declared per query and their dependencies drive scheduling. *Evidence chain*: the answer is tied to the passages, entities and paths that produced it.

3 METHOD

3.1 PROBLEM SETTING

We study retrieval over a scientific corpus $\mathcal{D}$ whose questions may require evidence from several documents and from more than one kind of evidence view. Indexing yields one shared set of dense indexes over text chunks, entities and typed relationships,

$$\mathcal{I} = \big(\mathbf{I}_{\mathcal{C}},\, \mathbf{I}_{\mathcal{V}},\, \mathbf{I}_{\mathcal{E}}\big), \tag{1}$$

whose construction, alias resolution and incremental assembly are described in Appendix A.5. Given a question q and the tool set $\mathcal{T} = \{\text{NAIVE}, \text{LOCAL}, \text{GLOBAL}\}$ of a configuration, MODERAG plans retrieval tasks, executes the tasks whose declared dependencies are ready, and reports an answer from the evidence they accumulate:

$$\mathcal{P} = \pi(q, \mathcal{T}), \qquad \mathcal{H} = \xi(\mathcal{P}; \mathcal{I}), \qquad a = \rho(q, \mathcal{H}), \tag{2}$$

where π, ξ, and ρ are the Planner, Executor, and Reporter: prompted agents built on a frozen LLM, for which we write θ as a model label where a symbol is needed (e.g., the keyword extractor g_θ) but train no parameters of our own. The Planner writes the plan as a set of sub-tasks,

$$t_i = (\mathrm{id}_i, m_i, q_i, \mathrm{dep}_i), \qquad m_i \in \mathcal{T}, \tag{3}$$

where id_i identifies the task, m_i is the retrieval mode assigned to sub-query q_i, and dep_i holds the identifiers of its prerequisites. Each task invokes exactly one retrieval operator on the shared index,

$$\mathcal{R}_i = \psi_{m_i}(q_i; \mathcal{I}), \qquad m_i \in \mathcal{T}, \tag{4}$$

whose raw output $\mathcal{R}_i$ we distinguish from the evidence $\mathcal{S}_i$ the task contributes after fusion. The mode is selected per sub-query rather than once per question, so a multi-faceted question yields tasks in several modes while a single-source question stays on dense retrieval.

3.2 PLANNING AND EXECUTION

Let $\mathcal{P} = \{t_1, \ldots, t_n\}$ be the plan returned by the Planner. Its dependency structure is a directed graph on the tasks,

$$A = \big\{(t_j, t_i) : t_j \in \mathrm{dep}_i\big\}, \qquad G_{\mathcal{P}} = (\mathcal{P}, A), \tag{5}$$

and a well-formed plan admits a consistent ordering of its tasks

$$\exists\, \sigma : \mathcal{P} \to \{1, \ldots, n\} \quad \text{such that} \quad t_j \in \mathrm{dep}_i \;\Rightarrow\; \sigma(t_j) < \sigma(t_i). \tag{6}$$

Equations 5 and 6 define the plan interface. The Planner is prompted to satisfy it: the tasks carry the dependency relation, and the returned execution order is required to be one of its topological orders.

Algorithm 1 ModeRAG query processing

Require: query q, tool set $\mathcal{T}$, index $\mathcal{I} = (\mathbf{I}_\mathcal{C}, \mathbf{I}_\mathcal{V}, \mathbf{I}_\mathcal{E})$
Ensure: answer a and evidence chain $\mathcal{H} = (\mathcal{S}, \mathcal{Z}, \mathcal{G}_q)$

$\mathcal{P} \leftarrow \pi(q, \mathcal{T})$ {Planner: sub-queries, modes, dependencies (Eqs. 3 and 5)}
apply the fallback $\sigma(\cdot)$ to rewrite unavailable modes
$U \leftarrow \mathcal{P}$; $C \leftarrow \emptyset$; $\mathcal{H} \leftarrow \emptyset$
while $U \neq \emptyset$ **do**
  $B \leftarrow \{t_i \in U \mid \mathrm{dep}_i \subseteq C\}$ {ready batch}
  if $B = \emptyset$ **then**
    break {cyclic plan or failed prerequisites}
  end if
  run B with at most W_{par} workers
  for each task $t_i \in B$ **do**
    $\mathcal{R}_i \leftarrow \textsc{RunTool}(m_i, q_i; \mathcal{I})$, up to r retries {raw evidence}
    if naive tool available and $m_i \neq$ NAIVE **then**
      $\mathcal{R}_i \leftarrow \mathcal{R}_i \uplus \textsc{RunTool}(\textsc{naive}, q_i; \mathcal{I})$ {All Modes: one dense pass per graph task}
    end if
    $\mathcal{S}_i \leftarrow \mathrm{Cap}_B(\mathcal{R}_i)$ {fusion, Eq. 10}
    merge $\mathcal{S}_i$ into $\mathcal{S}$ and its metadata (thread-safe)
    if t_i succeeded **then**
      $C \leftarrow C \cup \{t_i\}$
    end if
    $U \leftarrow U \setminus \{t_i\}$
  end for
end while
$a \leftarrow \rho(q, \mathcal{H})$ {Reporter: answer synthesis}
return a and $\mathcal{H}$

The Executor does not verify acyclicity. A plan that violates the interface is not rejected; it simply stops making progress, which we use as the termination condition below.

Algorithm 1 states the execution procedure. At each round it dispatches every pending task whose prerequisites have completed successfully, runs that ready batch concurrently with at most W_{par} workers, retries a failed task up to r times, and accumulates the evidence each task returns. Independent tasks finish even when another task fails, and the procedure terminates on every input: if no task is ready while tasks remain pending, the signature of a cyclic plan or of failed prerequisites, the Executor abandons the remainder and the Reporter answers from the evidence collected so far (Appendix A.4.2). A deterministic plan-time fallback rewrites modes unavailable in a particular ablation, but does not reroute run-time failures. Dependencies order execution and nothing else. They let independent sub-queries run in parallel and keep an unreliable hop from rewriting later sub-queries, at the cost that a later sub-query cannot be reformulated from an earlier retrieval result. Appendix A.8 derives the number of rounds, which depends on the width of the plan and the depth of its dependency structure as well as on the worker limit.

3.3 RETRIEVAL TOOLS

The three tools query the shared index for text chunks, entities, and typed relationships, and each supplies a complementary evidence view:

Table 2: The three retrieval modes available to the Planner.

Mode	Retrieval operation	Evidence need
Naive	Dense search over text chunks	Local facts and values
Local	Entity recall and one-hop graph traversal	Entity-centered links
Global	Relationship recall and cross-document synthesis	Broad comparisons

Every operator has the same recall-then-select shape but differs in the objects it recalls and in the graph operations it applies:

$$\textbf{Naive:} \quad \mathcal{R}_{\text{NAIVE}}(q) = \text{TopK}_{k_2}\big(\text{Dense}\big(q, \mathbf{I}_{\mathcal{C}}; k_1\big)\big), \tag{7}$$

$$\textbf{Local:} \quad \mathcal{R}_{\text{LOCAL}}(q) = \Xi_{\text{ent}}\big(\mathcal{K}^{(l)}, \mathcal{I}\big), \tag{8}$$

$$\textbf{Global:} \quad \mathcal{R}_{\text{GLOBAL}}(q) = \Xi_{\text{rel}}\big(\mathcal{K}^{(h)}, \mathcal{I}\big), \tag{9}$$

where $\text{Dense}(q, \mathbf{I}; k)$ denotes an approximate-nearest-neighbour recall of k items from index $\mathbf{I}$ and Ξ_{ent}, Ξ_{rel} are the entity- and relationship-centric expansions of Appendix A.3. Naive retrieval embeds the sub-query and returns the k_2 highest-ranked passages of the chunk index. Local retrieval starts from low-level keywords, recalls candidate entities, expands their incident one-hop edges, and ranks those edges by endpoint degree and accumulated extraction weight before collecting the source chunks of the retained entities. Global retrieval works at the relationship level: it recalls relationships from the relationship index while preserving their semantic order and collects the chunks that document them. Neither graph mode is seeded by a dense query, and both return a compact relation view together with the passages that support it. The keyword extractor, the recall depths (k_1, k_2, k_v, k_r), the edge ranking, the chunk selection rule and the tool prompts are specified in Appendix A.3.

In All Modes, each planned local or global task additionally runs the naive pass on its own sub-query, and the raw evidence of the two modes is merged under a fixed cap:

$$\mathcal{S}_i = \text{Cap}_B\big(\mathcal{R}_{m_i} \uplus \mathcal{R}_{\text{NAIVE}}(q_i)\big), \qquad |\mathcal{S}_i| \le B, \tag{10}$$

where $\uplus$ deduplicates by content while keeping the provenance label of each item, and $\text{Cap}_B(\cdot)$ ranks the union by query–chunk cosine similarity, places the entity and relation listings returned by the graph modes below quotable passages, and retains the B highest-ranked items. Every passage therefore carries a score on one scale regardless of the mode that produced it, which makes the merge meaningful rather than an artefact of insertion order. Graph-only ablations do not register the naive tool and skip Eq. 10.

3.4 Evidence chain

The evidence chain spans the whole pipeline, from the Planner's task decomposition through each Executor's retrieval results to the Reporter's synthesis, so that provenance is recorded as tools execute rather than reconstructed after the answer is written. It is the triple

$$\mathcal{H} = (\mathcal{S}, \mathcal{Z}, \mathcal{G}_q), \tag{11}$$

where $\mathcal{S}$ collects the retrieved evidence items, each annotated with its content, mode, score, producing task and source document; $\mathcal{Z}$ records task-level outputs; and $\mathcal{G}_q$ stores the retrieved graph entities, edges and paths. Because evidence items are carried as labeled items rather than merged by content, the mapping from the Reporter's context back to the task and retrieval mode that produced it stays explicit, which supports post-hoc verification, debugging and downstream trust calibration across a multi-agent pipeline rather than within a single retrieval call. The Reporter conditions on the original query and on $\mathcal{H}$. Our experiments use concise mode, which answers directly from the collected context and reports when it is insufficient. An optional reasoning mode attributes claims to evidence items and attaches confidence scores; its output schema is described in Appendix A.4.3.

4 Experiments

Benchmarks and protocol. We evaluate on two benchmarks. `LHCpubs-100QA` contains 100 questions, 25 per genotype, written over a 96-paper *indexed subset* of our released CMS/ATLAS corpus (Section B); every question requires evidence from at least two papers. All retrieval experiments run on that indexed subset, while the corpus is released in full (2,771 full-text papers, Table 3) so that the pipeline can be re-run on other subsets. `AI4EIC2023` provides 50 mostly single-paper questions over 38 workshop papers. The LHC questions cover complementary bridges (T1), cross-paper aggregation (T2), heterogeneous hops (T3), and three-retriever questions (T4). Both benchmarks use the same knowledge-graph indexer, DeepSeek-V4-Flash answer/planning backend, BAAI/bge-m3 embeddings, and Ragas metrics (faithfulness, answer relevancy, and correctness). Full settings and the paired bootstrap protocol are in Appendix B.

Table 3: The released `LHCpublications` corpus. Records come from INSPIRE; those without an arXiv identifier are listed with their DOI rather than dropped. All 2,338 papers they reference appear in the corpus, which holds 2,771 converted papers in total, including 433 further LHC documents. Curation removed 58 Herschel-ATLAS papers matched on the name and four unrelated documents.

Source	Metadata records	With arXiv id	Without arXiv id
ATLAS	1,677	1,560	117
CMS	1,912	1,633	279
Total	3,589	3,193	396

Table 4: Indexed subsets and question sets. Both subsets share one indexing pipeline, with 512-token chunks and 50-token overlap; the counts describe the indexed subsets, not the released corpus. All `LHCpubs-100QA` questions require multi-paper evidence; `AI4EIC2023` questions are mostly single-paper.

Benchmark	**Papers**	**Chunks**	**Entities**	**Relations**	**QA**
`AI4EIC2023`	38	1,083	11,302	20,064	50
`LHCpubs-100QA`	96	4,772	24,909	61,874	100

The four LHC genotypes distinguish the evidence structure of a question. T1 asks for a complementary bridge between two papers, where one paper supplies an entity or value needed to interpret the other. T2 aggregates a shared measurement across papers. T3 combines heterogeneous hops, such as a passage-level fact, an entity link, and a comparison. T4 requires a concrete value plus two additional retrieval steps of different kinds. Questions were retained through a partial retrievability gate: after normalizing eligible gold snippets, at least half had to match a vector top-20 chunk or have their cited paper represented in the retrieved metadata. The gate confirms that some annotated evidence is reachable at corpus level; it does not require every supporting passage to fit in one query's generation context. Construction details are in Appendix C.

4.1 Main results

This subsection and the next three report one evaluation each: Table 5 compares ModeRAG against external systems, Table 6 varies the tool set exposed to the Planner while holding the planning and reporting framework fixed, Table 7 accounts for the orchestration cost of each configuration, and Table 8 breaks the LHC result down by evidence genotype.

Answer Correctness compares the response with the reference answer, Answer Relevancy measures whether it addresses the query, and Faithfulness measures consistency with the retrieved context. The same DeepSeek backend generates answers and computes the Ragas metrics.

Table 5: **Main results across systems.** Means over questions; **C**: answer correctness, **R**: answer relevancy, **F**: faithfulness; ↑: larger is better. All systems answer the same questions from the same corpus; **bold** and underline mark the best and second-best value per column.

	LHCpubs-100QA			AI4EIC2023		
System	C ↑	R ↑	F ↑	C ↑	R ↑	F ↑
Vanilla RAG (Lewis et al., 2020)	0.4321	0.5156	0.5487	0.5418	0.6032	0.8124
GraphRAG (Edge et al., 2024)	0.5217	0.5583	0.6019	0.5364	0.6541	0.8836
LightRAG (Guo et al., 2024)	0.5794	<u>0.6075</u>	**0.6851**	<u>0.5716</u>	0.6907	<u>0.9548</u>
Agentic Hybrid RAG (Jiang et al., 2026)	<u>0.6042</u>	0.5928	0.6385	0.5537	<u>0.7084</u>	0.9513
ModeRAG (ours)	**0.6685**	**0.6110**	<u>0.6698</u>	**0.5840**	**0.7209**	**0.9721**

The single-call Vanilla RAG baseline and ModeRAG are our own implementations; Microsoft GraphRAG (Edge et al., 2024), LightRAG (Guo et al., 2024) and Agentic Hybrid RAG (Jiang et al., 2026) are run as released, on the same questions over the same corpus. For LightRAG we report

two configurations: its native hybrid mode, which combines local and global graph retrieval and is the configuration its own evaluation reports as the full system, and its graph-plus-vector mix mode, which matches the tool set available to ModeRAG. The internal rows of Table 6 map onto these configurations: naive corresponds to a naive-only setting, local and global to the two single-level graph modes, and local+global to a hybrid configuration; ModeRAG is the full tool set.

4.2 TOOL-SET ABLATION

Naive Only uses the same agent pipeline but restricts its tool set to dense passage retrieval. Local Only and Global Only restrict it to a single graph mode, and the remaining rows of Table 6 expose every other subset of the three modes; the graph-only configurations do not have the naive vector tool registered. All Modes exposes the complete tool set and adds vector fusion to each planned graph-mode task, so it runs a dense-vector pass for every graph task and the table does not isolate the graph contribution at a fixed vector-call budget. The table holds the planning and reporting framework fixed while varying the retrieval capabilities available to the Planner; it does not hold the number of vector calls or the total context volume fixed.

Table 6: **Tool-set ablation of ModeRAG.** Each row exposes a subset of the three retrieval modes to the same Planner–Executor–Reporter pipeline; **C**: answer correctness, **R**: answer relevancy, **F**: faithfulness; ↑: larger is better. **Bold** and underline mark the best and second-best value per column.

	LHCpubs-100QA			AI4EIC2023		
Tools enabled	C ↑	R ↑	F ↑	C ↑	R ↑	F ↑
naive	0.6016	0.5444	**0.6970**	<u>0.5572</u>	**0.6682**	<u>0.9436</u>
local	0.5044	0.4037	0.6382	0.5111	0.7156	0.9255
global	0.5323	0.4248	0.6407	0.4365	0.5350	0.8979
naive+local	<u>0.6212</u>	<u>0.5718</u>	<u>0.6743</u>	0.5407	<u>0.7421</u>	0.9518
naive+global	0.6087	0.5581	0.6516	0.5294	0.7336	0.9452
local+global	0.5231	0.4961	0.5667	0.5005	0.6340	0.9061
naive+local+global (**ModeRAG**)	**0.6685**	**0.6110**	0.6698	**0.5840**	0.7209	**0.9721**

Exposing all three modes improves answer correctness over Naive Only: the paired difference is $+0.077$ (95% paired bootstrap CI $[+0.042, +0.114]$) on `LHCpubs-100QA` and $+0.027$ (95% CI $[-0.017, +0.081]$) on `AI4EIC2023`, where it remains unresolved at $n = 50$. Table 9 gives the paired estimates for these headline gaps.

4.3 COST OF ORCHESTRATION

The dependency structure is used rather than decorative: 29 of the 100 `LHCpubs-100QA` plans contain at least one dependency, the realized schedule averages 1.31 rounds (maximum 3), and in the released run no plan was abandoned and no task needed a retry. On `AI4EIC2023` every plan is a single round. Each round dispatches the whole set of pending tasks whose dependencies have completed, so the round count grows with the depth of the dependency structure rather than with the number of modes alone. The nominal call counts implied by a plan are derived in Appendix A.7; the quantities below are the realized ones.

On `LHCpubs-100QA`, All Modes averages 3.25 planned sub-tasks over 1.31 rounds and uses 24,268 tokens and 86.7 s per question; graph-task vector fusion is included in that cost, and Local+Global has the largest measured cost (27,601 tokens, 125.9 s). On `AI4EIC2023`, All Modes averages 1.44 planned sub-tasks in a single round and uses 6% more tokens than Naive Only. The full configuration adds vector retrieval to each graph task, so the table supports dense retrieval as a useful component of this implementation but does not isolate graph-evidence gains under matched retrieval calls or a matched context budget. Full cost accounting is in Appendix B.

4.4 EVIDENCE-DEMAND GENOTYPES

The Planner issued more than one retrieval mode for 85 of the 100 `LHCpubs-100QA` questions, averaging 3.25 tasks per question (up to seven); on `AI4EIC2023` it did so for only 7 of 50, with 34

Table 7: **Cost of orchestration. Subtasks**: mean Planner sub-tasks per question; **Rounds**: mean Executor rounds, i.e. dependency layers executed, per question (Section A.4.2); **Tokens**: mean LLM tokens per question, over planning, keyword extraction and reporting; **Walltime**: mean wall-clock seconds per question. ↓: lower is better, with **bold** and underline marking the cheapest and second-cheapest component per column within a block; tied columns are unmarked. Rows are the components of Table 6; All Modes is ModeRAG.

Benchmark	Component	Subtasks ↓	Rounds ↓	Tokens ↓	Walltime (s) ↓
LHCpubs-100QA	Naive Only	2.71	**1.09**	**18,698**	**75.0**
	Local Only	2.66	**1.09**	20,495	97.8
	Global Only	**2.55**	1.14	20,156	95.5
	Local+Global	3.41	1.31	27,601	125.9
	All Modes	3.25	1.31	24,268	86.7
AI4EIC2023	Naive Only	1.40	1.00	**7,442**	**33.8**
	Local Only	**1.36**	1.00	7,505	34.5
	Global Only	1.44	1.00	7,575	35.1
	Local+Global	1.40	1.00	8,558	39.7
	All Modes	1.44	1.00	7,921	36.4

questions receiving a single task. Table 8 breaks the ModeRAG result down by genotype: T1 and T4 carry positive paired estimates ($+0.093$, 95% CI $[+0.015, +0.181]$; and $+0.054$, $[+0.003, +0.108]$), T2 is the largest ($+0.130$), and T3 is unresolved ($+0.030$, CI crossing zero). The differences track how much cross-paper evidence each specification requires.

Table 8: **Query genotypes on LHCpubs-100QA** ($n = 25$ per genotype), for ModeRAG. **Multi-mode**: fraction of questions with more than one retrieval mode; **C/R/F**: answer correctness, relevancy and faithfulness, ↑ larger is better, with **bold** and underline marking the best and second-best value per column.

Genotype	Multi-mode	C ↑	R ↑	F ↑
T1 complementary bridge	80%	0.5584	0.5043	0.6588
T2 cross-document aggreg.	64%	0.6998	0.6607	**0.7962**
T3 heterogeneous multi-hop	96%	0.6751	**0.6884**	0.6432
T4 three-retriever	100%	**0.7408**	0.5906	0.5809

5 Conclusion

We presented ModeRAG, a Planner–Executor–Reporter framework that treats dense retrieval, entity-centred graph traversal and relationship-centred synthesis as interchangeable tools, and we release the LHCpublications corpus together with LHCpubs-100QA, 100 multi-paper questions over a 96-paper indexed subset, and a 50-question mostly single-paper benchmark for contrast. Planning over the three modes improves answer correctness by $+7.7$ points over a prompt-matched single-mode pipeline on the multi-paper benchmark (95% paired bootstrap CI $[+0.042, +0.114]$), while the difference on the mostly single-paper benchmark is unresolved; restricting the same pipeline to graph retrieval stays below the prompt-matched vector baseline, so graph retrieval augments a dense-retrieval backbone here rather than replacing it. The full configuration reaches that result at 3.25 planned sub-tasks, 1.31 execution rounds, 24,268 tokens and 86.7 seconds per question, which makes the cost–accuracy trade-off part of the contribution rather than a footnote to it.

These findings are specific to two benchmarks in one scientific domain, one LLM backend and one embedding model, with prompted rather than trained planning and with generation and judging sharing a backend. They motivate controlled evaluation of orchestration on further corpora, of matched retrieval-call and context budgets, and of routers that decide when a single retrieval suffices, rather than a general claim that more retrieval modes always help.

REPRODUCIBILITY STATEMENT

The appendix documents the corpus and benchmark construction, model and retrieval settings, and evaluation protocol. We release the `LHCpublications` corpus (INSPIRE metadata, full-text markdown, a manifest covering every corpus paper, the 396 records that carry no arXiv identifier, and prebuilt graph and vector indexes over the 96-paper indexed subset), the 100-question `LHCpubs-100QA` benchmark, and the scripts and prompts that build both (`build_corpus.py`, `gen_qa.py`, `qa_gate.py`, `compose_benchmark.py`). Retrieval experiments are driven by the indexing and benchmark harness described in Appendix B.

AI USE STATEMENT

Generative AI tools were used to aid and polish writing, retrieve and discover related work, support research ideation and execution, draft sections of the paper, and generate synthetic datasets. Details are described in the paper. The authors reviewed the AI-assisted material and take responsibility for the final content, including text, claims, and artifacts produced with AI assistance.

REFERENCES

Jinheon Baek, Alham Fikri Aji, and Amir Saffari. Knowledge-augmented language model prompting for zero-shot knowledge graph question answering. *arXiv preprint arXiv:2306.04136*, 2023.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216*, 2024.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*, 2021.

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

Shahul Es, Jithin James, Luis Espinosa Anke, and Steven Schockaert. Ragas: Automated evaluation of retrieval augmented generation. In *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 150–158, 2024.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels. *arXiv preprint arXiv:2212.10496*, 2022.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. Lightrag: Simple and fast retrieval-augmented generation. *arXiv preprint arXiv:2410.05779*, 2024.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Su, et al. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. *arXiv preprint arXiv:2402.07630*, 2024.

Ruobing Jiang, Dawei Fu, Cheng Jiang, Tianyi Yang, Zijian Wang, Youpeng Wu, Yong Ban, Yajun Mao, and Qiang Li. Agentic hybrid rag for evidence-grounded muon collider analysis, 2026. URL `https://arxiv.org/abs/2606.10381`.

LangChain. Langgraph: Building stateful, multi-actor applications with llms. https://github.com/langchain-ai/langgraph, 2024.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33: 9459–9474, 2020.

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

Alireza Salemi and Hamed Zamani. Evaluating retrieval quality in retrieval-augmented generation. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2395–2400, 2024.

Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. Agentic retrieval-augmented generation: A survey on agentic RAG. *arXiv preprint arXiv:2501.09136*, 2025.

Viju Sudhi, Sinchana Ramakanth Bhat, Max Rudat, and Roman Teucher. Rag-ex: A generic framework for explaining retrieval augmented generation. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2776–2780, 2024.

Yixuan Tang and Yi Yang. Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries. *arXiv preprint arXiv:2401.15391*, 2024.

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

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. In *Proceedings of the First Conference on Language Modeling (COLM)*, 2024.

Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. Corrective retrieval augmented generation. *arXiv preprint arXiv:2401.15884*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

APPENDIX

This appendix collects detailed method and implementation notes, the extended experimental protocol and diagnostics, benchmark construction notes, and additional limitations.

A DETAILED METHOD AND IMPLEMENTATION

In this section, we formally describe MODERAG. We first introduce the notation and the problem formulation, then detail the three retrieval tools, the multi-agent pipeline, the knowledge graph construction procedure, and finally the evidence chain model together with a complexity analysis.

A.1 Notation and Problem Formulation

We denote scalars as lowercase letters (e.g., k), vectors as boldface lowercase letters (e.g., $\mathbf{e}$), and sets and graphs as calligraphic letters (e.g., $\mathcal{G}, \mathcal{E}$). The input corpus is a collection of documents $\mathcal{D} = \{d_1, \ldots, d_{|\mathcal{D}|}\}$. After chunking, the corpus is partitioned into a set of chunks $\mathcal{C} = \{c_1, \ldots, c_{|\mathcal{C}|}\}$, each annotated with its source document. A knowledge graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is constructed over the corpus, where $\mathcal{V}$ is the entity set, $\mathcal{R}$ is the set of relation types, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{R} \times \mathcal{V}$ is the set of typed directed edges (s, r, o); each edge $e \in \mathcal{E}$ carries an accumulated weight $w(e)$ and the chunk provenance $\mathrm{chunk_ids}(e)$ from which it was extracted. We maintain three dense indexes built with a shared embedding function $\phi(\cdot)$: a chunk index $\mathbf{I}_\mathcal{C}$, an entity index $\mathbf{I}_\mathcal{V}$, and a relationship index $\mathbf{I}_\mathcal{E}$, all queried via ANN search.

Following Edge et al. (2024); Guo et al. (2024), we define the GraphRAG task as answering a natural-language query q by first retrieving a question-specific evidence set from $\mathcal{G}$ and the chunk index, and then generating an answer grounded in that evidence. Our system solves this task through the Planner–Executor–Reporter pipeline of Section 3 (Eq. 2), whose operators we instantiate below.

The Reporter receives the original query as well as the evidence chain, and the remainder of this section instantiates each operator.

A.2 System Overview

Given a query q, the system proceeds through three stages (Algorithm 1).

Planning. The Planner, an LLM-based agent, decomposes q into a set of sub-tasks $\mathcal{P} = \{t_1, \ldots, t_n\}$. Each sub-task is the tuple of Eq. 3,

where $\mathrm{id}(t_i)$ is a unique identifier, $m_i \in \mathcal{T} = \{\textsc{naive}, \textsc{local}, \textsc{global}\}$ is the assigned retrieval mode (i.e., the tool to invoke), q_i is the sub-query derived from q, and $\mathrm{dep}(t_i) \subseteq \mathcal{P} \setminus \{t_i\}$ is the dependency set. The Planner is prompted to emit an acyclic dependency structure together with a consistent execution order, but the pipeline does *not* verify acyclicity: there is no cycle detection on $\mathrm{dep}(\cdot)$, and the Executor discovers ready tasks lazily rather than by a topological sort (Section A.4.2). A cyclic plan is therefore not rejected; the Executor exhausts the set of ready tasks, emits a warning, and skips the remainder, so the Reporter synthesizes its answer from partial evidence. The DAG requirement is thus a property we ask the Planner to respect, not an invariant the system enforces. The Planner is constrained to use only the modes available in the tool set $\mathcal{T}$, and applies a deterministic fallback mapping $\sigma : \mathcal{T} \to 2^\mathcal{T}$ (with priorities $\textsc{naive} \to [\textsc{local}, \textsc{global}]$, $\textsc{local} \to [\textsc{naive}, \textsc{global}]$, and $\textsc{global} \to [\textsc{local}, \textsc{naive}]$) to rewrite any unavailable mode before the plan is accepted.

Execution. The Executor repeatedly selects pending tasks whose dependencies have completed successfully and invokes their retrieval tools, collecting the returned evidence. Tasks in the same ready set are submitted together to a bounded thread pool. A dependency is a scheduling constraint only: the Planner has already fixed every sub-query, and retrieved evidence from a completed task is not supplied to downstream task queries. Failed tasks are retried up to R additional times, and independent tasks continue in the presence of partial failures.

Reporting. The Reporter synthesizes the final answer a from the aggregated evidence chain $\mathcal{H}$, either in a concise form or with an explicit step-by-step reasoning chain that attributes each claim to its supporting evidence.

A.3 Retrieval Tools

The three retrieval modes are implemented as tools registered in a central `ToolRegistry`. Each mode shares a two-stage recall-then-select structure but differs in the objects it retrieves (chunks, entities, or relationships) and in the graph operations it performs. Every returned chunk is annotated with its query–chunk similarity,

$$\mathrm{score}(c) = \cos\big(\phi(q), \phi(c)\big), \tag{12}$$

computed with the same embedding function $\phi(\cdot)$ for every mode. The graph modes choose which chunks to return structurally, by the association counts described below, and then report this score

for each of them; the fusion in Eq. 10 ranks on this single scale. Let k_1 denote the recall depth and k_2 the number of chunks returned to the downstream modules (we use $k_1 = 20$, $k_2 = 5$). Each tool optionally supports the cross-encoder re-ranker `BAAI/bge-reranker-v2-m3` for query-side re-ranking; it is disabled in the reported experiments. We do not report a re-ranker ablation in this version.

A.3.1 NAIVE MODE

Naive mode performs pure dense retrieval without graph traversal. Given the query q, it recalls the top-k_1 chunks via ANN search on $\mathbf{I}_{\mathcal{C}}$ and returns the top-k_2 of them (Eq. 7).

This mode is best suited for factual and numeric queries whose answer lies within individual chunks (e.g., beam energies, luminosity, detector parameters).

A.3.2 LOCAL MODE

Local mode is designed for entity-centric queries that require traversing relationships around specific entities. The retrieval process proceeds in five steps.

(i) Low-level keyword extraction. The LLM-based keyword extractor $g_\theta^{(l)}(\cdot)$ produces low-level keywords (entities, proper nouns, technical terms),

$$\mathcal{K}^{(l)} = g_\theta^{(l)}(q), \qquad \mathcal{K}^{(h)} = g_\theta^{(h)}(q), \tag{13}$$

where $\mathcal{K}^{(h)}$ collects the high-level concepts and themes that global mode consumes.

(ii) Entity recall and degree re-ranking. Candidate entities are recalled by ANN search on $\mathbf{I}_{\mathcal{V}}$, then re-ranked by their degree centrality in the graph,

$$\mathcal{V}_0 = \mathrm{TopK}_{\mathbf{I}_{\mathcal{V}}}\big(\mathcal{K}^{(l)}, k_v\big), \qquad \mathcal{E}_v = \big\{(s,r,o) \in \mathcal{E} : s \in \mathcal{V}_0 \vee o \in \mathcal{V}_0\big\}, \tag{14}$$

where $\mathcal{E}_v$ is the incident one-hop edge set expanded in step (iii).

k_v is the entity recall depth (library default $k_v = 10$; our experiments run $k_v = 3$, Section 4). All recalled entities are retained for expansion; degree sorting orders this candidate set and does not itself prune entities.

(iii) Edge expansion. From each top entity we expand the one-hop neighborhood, collecting all incident edges, as in Eq. 14.

(iv) Dual-sort. Edges are ranked by the pair of endpoint-degree sum and accumulated extraction weight, compared lexicographically in descending order,

$$e \succ e' \iff \mathrm{rank}_{\mathrm{local}}(e) >_{\mathrm{lex}} \mathrm{rank}_{\mathrm{local}}(e'), \qquad \mathrm{rank}_{\mathrm{local}}(e) = \big(\deg(s)+\deg(o),\, w(e)\big), \tag{15}$$

where $\deg(e) = \deg(s) + \deg(o)$ is the sum of the endpoint degrees and $w(e)$ is the accumulated edge weight.

This balances structural prominence (high degree) with evidential strength (high co-occurrence weight). The implementation retains at most the top-k_r edges under this ordering ($k_r = 10$ in the benchmark harness); only these retained edges are serialized into the local graph context.

(v) Chunk collection and selection. Chunks are collected from the provenance of the selected entities, ranked by the number of associated entities, capped at 10 candidates per mode in the benchmark harness, and truncated to the final top-k_2 chunks, each scored by Eq. 12 (re-ranked by $\mathrm{rr}(\cdot)$ when enabled). The retained entities and edges are serialized into a graph context block passed to the Reporter.

A.3.3 GLOBAL MODE

Global mode is designed for cross-document synthesis and comparative analysis. Unlike local mode, which starts from entities, global mode operates at the relationship level.

(i) High-level keyword extraction. The extractor produces high-level keywords (concepts, themes, abstract ideas) as in Eq. 13.

(ii) Relationship recall. Relationships are recalled by ANN search on $\mathbf{I}_{\mathcal{E}}$, preserving cosine order,

$$\mathcal{E}_0 = \mathrm{TopK}_{\mathbf{I}_{\mathcal{E}}}\big(\mathcal{K}^{(h)}, k_r\big), \qquad \mathcal{V}_{\mathcal{E}_0} = \{s, o : (s, r, o) \in \mathcal{E}_0\}, \tag{16}$$

where $\mathcal{V}_{\mathcal{E}_0}$ collects the endpoints of the recalled relationships.

k_r is the relationship recall depth (library default $k_r = 5$; our experiments run $k_r = 10$, Section 4). No degree-based re-ranking is applied, so that the semantic ordering of the recall stage is retained.

(iii) Entity extraction. The endpoints of the recalled relationships are collected into the entity set $\mathcal{V}_{\mathcal{E}_0} = \{s, o \mid (s, r, o) \in \mathcal{E}_0\}$ without sorting or limiting.

(iv) Chunk collection and selection. Chunks are collected from the provenance of the recalled relationships, ranked by the number of associated relationships, capped at 10 candidates per mode in the benchmark harness, and truncated to the final top-k_2, each scored by Eq. 12 (re-ranked by $\mathrm{rr}(\cdot)$ when enabled). The endpoints of the recalled relationships and those relationships are serialized as a graph context block passed to the Reporter.

This mode is suited for broad topics, overviews, and comparative analyses where relevant facts are scattered across many relationships and documents.

A.4 Multi-Agent Pipeline

A.4.1 Planner

The Planner $\pi(\cdot)$ is a prompted LLM-based agent that maps a query and the available tool set to a structured plan $\mathcal{P} = \{t_i\}_{i=1}^{n}$ over the tuples of Eq. 3, with dependency graph $G_{\mathcal{P}}$ as in Eq. 5.

$m_i \in \mathcal{T}$ holds for all i. The Planner is also prompted to satisfy Eq. 6; the implementation does not verify it, and instead stops when a round has no ready task (Section A.4.2). The system prompt instructs the Planner to follow these heuristics:

- **Factual/numeric queries** ("What is X", beam energies, luminosity) → naive mode;
- **Entity-centric queries** (specific persons, detectors, experiments, facilities) → local mode;
- **Broad/comparative queries** (overviews, differences, themes) → global mode;
- **Multi-faceted queries** → multiple tasks with explicit dependencies.

For simple queries that require a single retrieval call, the Planner returns a plan with exactly one task, minimizing orchestration overhead. When the requested mode is unavailable, the deterministic fallback $\sigma(\cdot)$ rewrites the task type, guaranteeing that every plan is executable. We note precisely what σ does and does not cover. It repairs *mode availability*, and only that: it is applied once, at plan time, before the Executor sees the plan. It is unrelated to the Executor's handling of a tool that fails at run time, for which we implement *no* fallback; a task that exhausts its retry budget contributes no evidence and the pipeline proceeds with the rest (Section A.4.2). We state this explicitly because the two mechanisms are easy to conflate, and only the first exists in our implementation.

A.4.2 Executor

The Executor $\xi(\cdot)$ processes the plan with dependency-aware parallelism. It does not topologically sort $\mathcal{P}$; instead, at each round u it forms a batch from the pending tasks whose dependencies have completed successfully. Each batch is one layer of the dependency graph, dispatched as a barrier, and is counted as one round in Table 7. Let U_u be the pending tasks and C_u the set of tasks that completed successfully at the start of that round; the ready batch B_u is the batch selected in line 6 of Algorithm 1, and a round updates both sets,

$$U_{u+1} = U_u \setminus B_u, \qquad C_{u+1} = C_u \cup \big\{t_i \in B_u : \mathrm{ok}(t_i)\big\}, \tag{17}$$

with $\mathrm{ok}(t_i)$ true when the task returned evidence. Because each round (Eq. 17) consumes exactly one batch, the number of rounds is bounded below by the worker limit and by the longest dependency chain,

$$R_{\mathrm{rounds}} \;\geq\; \max\Big(\big\lceil n/W_{\mathrm{par}}\big\rceil,\; h(G_{\mathcal{P}})\Big), \tag{18}$$

where $h(G_{\mathcal{P}})$ is the length of the longest path in $G_{\mathcal{P}}$. This is a lower bound and not a characterisation: the realised number of rounds also depends on the width and topology of the plan, since a round can only dispatch the tasks whose prerequisites have already completed.

Tasks in B_u are submitted to a `ThreadPoolExecutor` with at most W_{par} workers; both the benchmark and the Executor default use $W_{\text{par}} = 4$. After the batch completes, successful tasks are added to C_u and all attempted tasks are removed from U_u. Evidence produced by the batch is merged into a single evidence chain under a lock. Failed tasks are retried up to $R = 1$ additional times; independent tasks continue despite partial failures. If $B_u = \emptyset$ while U_u is nonempty, the Executor records a warning and abandons the remaining tasks. There is no run-time fallback that rewrites a failed task onto a different mode (Section A.4.1), and completed-task evidence is not used to reformulate pending queries.

A.4.3 Reporter

The Reporter $\rho(\cdot)$ synthesizes the final answer from the evidence chain and supports two generation modes.

CONCISE mode. The Reporter generates a direct answer using the retrieved context, formatted as a compact response. The prompt strictly constrains the LLM to rely only on the provided context (knowledge graph entities, relationships, and document chunks), forbidding external knowledge, and to report insufficient evidence when the context does not suffice. Formally, $a = \rho^{\text{c}}(q, \mathcal{H})$.

REASONING mode. The Reporter first analyzes the evidence and produces an ordered chain of reasoning steps, then synthesizes the final answer with inline evidence citations:

$$\big\{(z_k, \mathcal{S}_k, \gamma_k)\big\}_{k=1}^{K},\ a \ = \ \rho^{\text{r}}(q, \mathcal{H}), \tag{19}$$

where z_k is a claim, $\mathcal{S}_k \subseteq \mathcal{H}$ is the set of evidence items supporting the claim, and $\gamma_k \in [0, 1]$ is a per-claim confidence score. Claims with $\gamma_k < 0.8$ are flagged with an explicit uncertainty note. Each evidence item is additionally linked back to the task that produced it, forming the evidence traces described in Section A.6.

A.5 Knowledge Graph Construction

The knowledge graph is built from the document corpus through a multi-stage pipeline.

A.5.1 Document Chunking

Documents are split into chunks using a token-based splitter with configurable chunk size L_c (default: 512 tokens) and overlap (default: 50 tokens). A truncation step removes trailing boilerplate (e.g., acknowledgments, references) to improve extraction quality.

A.5.2 Entity and Relationship Extraction

Each chunk is processed by an LLM prompt that extracts entities (name, type, description) and relationships (source, target, keywords, description) in a structured JSON schema via structured output decoding. The extractor supports parallel batch processing across chunks (up to 16 workers) as well as incremental extraction for newly added documents. Relationships are subject to N-ary decomposition into binary pairs and are described with high-level keywords for later global retrieval.

A.5.3 Entity Resolution

Extracted entities are deduplicated using an embedding-similarity + Union-Find procedure. Let $\mathbf{e}_v = \phi(v)$ be the embedding of entity name v. We build an HNSW index over all entity embeddings and define an adjacency relation $v \sim v'$ iff $\cos(\mathbf{e}_v, \mathbf{e}_{v'}) \geq \tau$, with threshold $\tau = 0.9$. Transitive closures of this relation, computed with Union-Find, group entity names into alias clusters, and each

cluster is collapsed to the canonical name with maximal frequency,

$$v \sim v' \iff \cos\big(\phi(v), \phi(v')\big) \geq \tau, \qquad c(v) = \arg \max_{v' \in \text{cluster}(v)} \text{freq}(v'), \tag{20}$$

$$w(e) = \sum_{\text{occurrences of } e} w_{\text{instance}}, \tag{21}$$

where $c(v)$ is the canonical form of an alias cluster and w_{instance} is a unit instance weight, so that repeated extraction of a relation accumulates its weight.

The frequency tie-breaking (favoring lexicographically larger names) ensures deterministic resolution. Merged entities concatenate their descriptions and accumulate the union of their chunk provenance.

A.5.4 Graph Assembly

Resolved entities form the node set of a directed graph maintained with `NetworkX`. Each extracted relationship becomes a directed edge (s, r, o), and repeated extractions of the same relationship accumulate its weight as in Eq. 21.

Each extracted relation is assigned $w_{\text{instance}} = 1$, so the accumulated weight is its extraction count. Nodes store their type, description, and chunk provenance; edges store their type, weight, keywords, and chunk provenance. These attributes are precisely the quantities consumed by the local and global retrieval modes in Section A.3.

A.6 Evidence Chain Model

The evidence chain is the system's primary mechanism for explainability and traceability. Formally, it is the triple of Eq. 11, where $\mathcal{S}$ is the ordered collection of retrieved evidence items, $\mathcal{Z}$ is the sequence of executor reasoning steps (per-task descriptions), and $\mathcal{G}_q$ aggregates graph metadata, including entities, edges, and traversal paths. Each item is the tuple

$$e = \big(\text{content},\ m,\ \text{score},\ \text{id},\ \text{src}\big), \tag{22}$$

recording the passage, the mode that produced it (naive/local/global), its relevance score, the task that produced it and its source document; evidence produced by task t_i is attributed to it,

$$\text{prov}(s) = \text{id}_i \quad \text{for every } s \in \mathcal{S}_i, \tag{23}$$

which keeps the mapping from the Reporter's context back to the task and mode that produced it explicit. In REASONING mode, the Reporter further attaches the reasoning chain $\{(z_k, \mathcal{S}_k, \gamma_k)\}$ and the evidence traces that map each evidence item to the claims it supports. This model supports both human inspection (for debugging and trust) and programmatic access (for downstream evaluation and analysis).

A.7 Orchestration Cost

The nominal number of retrieval operations follows from the plan. Writing n_n, n_l, and n_g for the number of naive, local, and global tasks,

$$C_{\text{LLM}} = 2 + n_l + n_g, \qquad C_{\text{dense}} = n_n + (n_l + n_g), \qquad C_{\text{graph}} = n_l + n_g. \tag{24}$$

The constant 2 counts the planning and reporting calls, each graph task adds one keyword-extraction call, C_{dense} counts passes over the chunk index (one per naive task and one per fused graph task), and C_{graph} counts the entity and relationship index searches performed by the graph operators. Equation 24 is nominal in two respects: each task may spend up to r retries, which can double the per-task terms, and reasoning mode adds one further reporter call. Orchestration cost therefore grows with the size and mode mix of the plan and with the depth of its dependency structure (Eq. 5), not with the number of modes alone. Section B.3 reports the measured calls, tokens, and wall-clock time per configuration alongside accuracy.

A.8 Complexity Analysis

We analyze the complexity of ModeRAG in two phases, following the analysis of LightRAG (Guo et al., 2024).

Indexing phase. Entity and relationship extraction requires one LLM call per chunk. For a document of T tokens, chunk length L_c, and overlap $O < L_c$, the number of chunks is 1 when $T \leq L_c$ and otherwise $1 + \lceil (T - L_c)/(L_c - O) \rceil$. These calls are executed concurrently by up to $W_e = 16$ workers. Embedding computation and ANN index construction add their usual costs; newly added documents can be processed incrementally and merged into the existing graph.

Retrieval phase. Each planned local or global task invokes one LLM keyword-extraction call, so a plan with n_l local and n_g global tasks spends $C_{\text{LLM}} = 2 + n_l + n_g$ calls on planning, extraction, and reporting, as in Eq. 24. The benchmark configuration also enables vector fusion: every local or global task runs a naive pass when that tool is available, which gives $C_{\text{dense}} = n_n + (n_l + n_g)$ passes over the chunk index on top of the $C_{\text{graph}} = n_l + n_g$ ANN searches that the graph operators perform over the entity and relationship indexes. Graph-only ablations do not register a naive tool and therefore skip the fusion passes. Each returned chunk additionally requires one batched embedding call to assign the similarity in Eq. 12, which is negligible next to the LLM calls but is included in the wall-clock measurements. The number of rounds, and hence the wall-clock time, follows Eq. 18: it grows with the plan size, the worker limit W_{par}, and the width and depth of $G_{\mathcal{P}}$.

B Extended Experimental Protocol and Diagnostics

We evaluate whether the configured Planner–Executor–Reporter system improves end-to-end answer quality over its internal retrieval ablations. We run both benchmarks under one protocol (Section B.1): the same indexer, backbone, and ragas judges, with ablations that differ in the tool set exposed to the Planner. In the All Modes configuration, graph tasks also receive a vector-fusion pass; consequently, the All Modes versus Naive Only contrast measures the full configured system and does not isolate graph evidence at a fixed number of vector calls. The two benchmarks differ in question composition (`LHCpubs-100QA` uses evidence from multiple papers, while `AI4EIC2023` is mostly single-paper and expository), but their differences do not establish a universal boundary for orchestration. We calibrate each paired per-question gap in Section B.1.

B.1 Experimental Setup

Benchmarks. `AI4EIC2023` is a corpus of 38 papers accepted to the AI4EIC2023 workshop on artificial intelligence for the Electron-Ion Collider, a next-generation nuclear-physics facility, covering jet physics, detector design, track reconstruction, particle identification, and ML applications. The 50 question–answer pairs are drawn from those papers; each carries a reference answer and the `arxiv_id` of its source paper. Most questions are single-paper and expository ("what are the advantages of ... and how are they related"), and the Planner decomposes only 6 of the 50 into more than two sub-tasks. That property turns out to be decisive (Section 4 and B.2).

Corpus construction. We collect ATLAS and CMS records from INSPIRE together with their arXiv identifiers: 3,589 metadata records (1,677 ATLAS, 1,912 CMS), 3,193 of which carry at least one identifier, referring to 2,338 distinct papers. The remaining 396 records have no arXiv identifier and are listed with their DOI in the release rather than dropped silently. Fetching and converting the sources yields 2,771 distinct markdown papers, so the corpus is broader than the metadata files: it contains all 2,338 papers they reference plus 433 further LHC documents, mainly conference proceedings, notes and early detector papers. Curation removed 62 papers that the collection had picked up by name rather than by content: 58 of the Herschel-ATLAS astronomy survey, which shares the name, and four unrelated documents. Each removal is listed with its reason in `removed_papers.json`, and the 57 INSPIRE records that referred only to removed papers are kept in `excluded_records.json`, so the removal is auditable. The release also includes `corpus_manifest.jsonl`, one row per corpus paper with its INSPIRE metadata; the 96-paper indexed subset used in every experiment contains none of the removed papers. `build_corpus.py` rebuilds the indexed subset either from a dataset clone or by downloading the markdown.

Benchmark construction. From the corpus we take 16 seed clusters and a 96-paper subset for indexing (Table 4), so chunks, entities and relations describe that indexed subset and not the full corpus. Questions are generated against four genotype specifications by `gen_qa.py`, filtered by the retrievability gate of Section C using `qa_gate.py`, and assembled into the released file by `compose_benchmark.py`. The released benchmark contains 100 questions, 25 per genotype, each with a gold answer and verbatim gold evidence spans; the annotated evidence of the released set is drawn from 80 of the 96 indexed papers:

- **T1, complementary bridge:** two papers where one supplies a value or entity that the other names but does not resolve; neither paper alone answers the question.
- **T2, cross-document aggregation:** the same measured quantity reported in two or more papers, requiring the values to be collected and compared.
- **T3, heterogeneous multi-hop:** different hops genuinely require different retrieval strategies (a verbatim detail, an entity-centred hop, and an aggregate).
- **T4, three-retriever mandatory:** the question cannot be answered without a concrete numeric/detail value *and* two further hops of a different kind.

Every candidate question passes a retrievability gate before it enters the benchmark. For each annotated evidence item with a snippet of at least 40 normalized characters, we take the first 80 characters beginning one quarter of the way into the snippet; an item counts as a hit if this fragment occurs in any vector top-20 chunk or if its cited paper identifier occurs in the metadata of any top-20 chunk. A question passes when at least half of its eligible evidence items are hits. Thus the gate establishes partial corpus-level reachability; it does not require one query to retrieve every supporting item into a generation context. The gate implementation is in the released benchmark scripts; Appendix C documents the question-set provenance.

Both indexed subsets are processed by the same indexing pipeline (Section A.5): documents are chunked (512-token chunks, 50-token overlap), entities and relationships are extracted into a knowledge graph with alias resolution at threshold $\tau = 0.9$, and dense indexes are built over chunks, entities, and relationships.

Evaluation metrics. We use the ragas framework (Es et al., 2024), whose metrics are computed per query and averaged over the benchmark: **Answer Correctness** (**C**), factual accuracy against the gold reference; **Answer Relevancy** (**R**), whether the answer addresses the query, penalising off-topic or extraneous content; and **Faithfulness** (**F**), whether the answer is factually consistent with the retrieved context. Relevancy and faithfulness are reference-free and probe grounding; correctness additionally verifies content against the reference. All three are judged by the same LLM backend that generates the answers.

Baselines and ablations. All configurations are produced by the same harness and share every implementation detail except the retrieval tool set exposed to the Planner:

- **RAG (Vector Only):** pure dense vector retrieval (recall depth $k_1 = 20$, final depth $k_2 = 5$) with no knowledge graph and no task decomposition, run once per question under the shared Reporter prompt.
- **Naive Only:** the full multi-agent pipeline restricted to the naive tool, i.e. orchestration over pure vector search. It isolates the pipeline from the graph modes.
- **Local Only / Global Only / Local+Global:** the full pipeline restricted to the entity-centric mode, the relationship-centric mode, or both without the naive mode.
- **All Modes:** the full system with all three modes available. In this configuration, each planned local or global task also receives a dense-vector evidence pass; these vector results are merged with the graph-mode results, up to the configured evidence cap.

When the Planner requests a mode outside the available tool set, the deterministic fallback $\sigma(\cdot)$ (Section A.4.1) rewrites it, so every configuration runs on every question.

Implementation details. All LLM operations use DeepSeek-V4-Flash (`deepseek-v4-flash-ioa`) through an internal OpenAI-compatible endpoint; embeddings are BAAI/bge-m3 (Chen et al., 2024); ragas is computed on the same backend. Unless stated otherwise: chunk size $L_c = 512$ tokens with 50-token overlap; recall and selection depths $k_1 = 20$, $k_2 = 5$; entity and relationship recall depths $k_v = 3$, $k_r = 10$, with a chunk-collection cap of 10 chunks per mode; thread-pool width $W_{\text{par}} = 4$; retry budget $r = 1$; entity resolution threshold $\tau = 0.9$. The Reporter runs in CONCISE mode throughout. Reported per-question LLM token totals sum every call made inside a query: planning, one keyword extraction per graph task, and reporting, including retries (Section B.1). The cross-encoder re-ranker is disabled for the reported agent and vector-only runs. The full configuration enables dense-vector fusion for graph-mode tasks; the graph-only ablations have no naive tool registered, so they execute without this fusion pass.

All configurations answer the same questions from the same corpus. The internal rows share one harness, including the Reporter prompt, so the internal contrast All Modes versus Naive Only measures the full-system difference under that harness, including the graph-task vector-fusion behavior described above. The external systems are run as released (Section 4.1), so their contrasts with ModeRAG compare complete systems rather than isolating a single design choice.

Noise floor. Every number we report is a mean over questions, so the unit of inference is the question, not the LLM call: $n = 50$ on `AI4EIC2023` and $n = 100$ on `LHCpubs-100QA`. We calibrate the resolution of a difference rather than assume one. The per-question difference in answer correctness between All Modes and Naive Only has standard deviation 0.18 on `LHCpubs-100QA`, giving a standard error of 0.018 and hence a 95% bootstrap interval of roughly ± 0.036 around the mean gap; on `AI4EIC2023` the same quantity has standard deviation 0.17, standard error 0.025, a ± 0.048 band. Table 9 states the headline gaps with their intervals, and we treat a gap whose interval contains zero as unresolved rather than as a null result. Two consequences are worth stating up front: a $+5$-point effect is *not* resolvable at $n = 50$ on this benchmark, and the three metrics are not equally noisy, so the same absolute gap can be resolved on one metric and not on another.

The genotype estimates vary. T2 is the largest positive estimate ($+0.130$, 95% CI $[+0.056, +0.221]$); T1 and T4 are also positive with intervals that exclude zero ($+0.093$, $[+0.015, +0.181]$; and $+0.054$, $[+0.003, +0.108]$). T3 ($+0.030$, CI crossing zero) and the single-paper benchmark do not resolve a correctness difference. These results do not establish that single-query retrieval is sufficient for every question in either group.

Table 9: **Headline gaps and their resolution.** Paired per-question differences (All Modes − Naive Only) with 95% bootstrap intervals over questions; W/L counts questions won and lost. Quality levels are in Table 5 and Table 6.

Benchmark	Comparison	Metric	Δ	95% CI	*n*	W/L
`LHCpubs-100QA`	All Modes − Naive Only	Correctness	$+0.077$	$[+0.042, +0.114]$	100	63/37
`AI4EIC2023`	All Modes − Naive Only	Correctness	$+0.027$	$[-0.017, +0.081]$	50	23/27

B.2 Ablation Study

The tool-set rows of Table 6 are the per-mode ablation: they share one Planner–Executor–Reporter pipeline and differ only in the tools registered with the Planner. Graph-only configurations have lower mean correctness than naive, while exposing all three modes gives the highest correctness and the highest answer relevancy. The current evidence therefore supports dense retrieval as a useful component in this implementation; it does not isolate the contribution of graph evidence at a matched vector-call or context budget. The Planner assigns 64% of its planned tasks to naive mode, but graph-task vector-fusion calls are additional to this planned-task count.

Faithfulness does not move with correctness. Within the tool-set ablation the naive-only configuration has the highest faithfulness (0.6970) and the lowest correctness among the multi-mode configurations, and across systems (Table 5) LightRAG is more faithful than ModeRAG on `LHCpubs-100QA` (0.6851 vs. 0.6698) while being less correct (0.5794 vs. 0.6685). Part of this is mechanical: faithfulness is a ratio over the claims an answer makes, so a system that retrieves

more and says more can be more correct and less faithful at the same time. The two comparisons also differ in scope, since the ablation rows are our own pipeline with subsets of the tools registered whereas Table 5 compares complete systems.

B.3 Efficiency Analysis

The pipeline has computational costs, and they increase with plan size. On `AI4EIC2023`, All Modes averages 1.44 planned sub-tasks in a single round and uses 6% more tokens than Naive Only. On `LHCpubs-100QA`, it averages 3.25 planned sub-tasks over 1.31 rounds and uses 24,268 tokens and 86.7 s per question. Graph-task vector fusion is included in All Modes cost. Local+Global has the largest measured cost (27,601 tokens, 125.9 s). The external baselines of Section 4.1 are the protocol under which mode-matched cost comparisons are to be made; here we report cost only for the internal configurations above.

B.4 Error Analysis

The failure modes we observe on `LHCpubs-100QA` cluster into four groups:

- **Under-decomposition:** the Planner returns a single sub-task for a question whose evidence is split across papers. This is the dominant failure on T3, where correctness is closest between configurations, and it is also the modal behaviour on `AI4EIC2023` (34 of 50 questions).
- **Entity resolution errors:** distinct entities merged, or the same entity left unmerged across papers (Section A.5.3). These propagate directly into the local mode, whose answers are the least faithful of the three modes.
- **Relevance drift in traversal:** later hops of a multi-hop local traversal attach to a tangentially related entity, adding context that is grounded but off-topic. This is the most likely source of the faithfulness gap in Section B.2.
- **Non-committal answers:** the ragas relevancy judge assigns 0 to any answer it classifies as non-committal, and 18 of the 100 All Modes answers on `LHCpubs-100QA` (and 8 of 50 on `AI4EIC2023`) receive such a zero. Many of these answers are correct in substance and honest about a gap in the corpus, so the reported relevancy averages are depressed by the metric rather than by the system.

B.5 Reproduction details

The experiments use `build_index.py` to construct the graph and vector indices, `run_benchmark.py` for agent configurations, and `run_benchmark_rag.py` for the vector-only baseline. These live in `benchmarks/common/` and are shared by both benchmarks, each of which exposes a `run.sh` that drives them and writes per-question scores into `results/`; the paired confidence intervals are recomputed by `analysis/bootstrap_ci.py`. The released `LHCpubs-100QA` directory additionally records the per-batch question-generation parameters. The environment requires `langchain-community < 0.4` for the ragas dependency.

C LHCpubs-100QA: Construction of the Question Set

This appendix documents how the `LHCpubs-100QA` question set was constructed. The indexed subset these questions are written over is 96 of the corpus papers, indexed by the pipeline of Section A.5. Questions were generated in batches (labelled `newgen:*` in the released file, with each question carrying its batch as provenance) against four genotype specifications, then filtered by a retrievability gate before any configuration was scored.

Retrievability gate. A question enters the benchmark if at least half of its eligible annotated evidence items pass the gate. We normalize snippets with Unicode NFKC and remove whitespace; snippets shorter than 40 characters are excluded. For each remaining snippet, we test whether its first 80 characters beginning at one quarter of the snippet occur in a vector top-20 chunk, or whether the cited paper identifier occurs in the metadata of any retrieved chunk. This is a partial corpus-level

reachability check: it does not require all supporting papers or spans to be present in one query's candidate set. The released benchmark contains 100 questions, 25 per genotype.

Genotype definitions. T1 (*complementary bridge*) requires two papers where one supplies a value or entity the other names but does not resolve; T2 (*cross-document aggregation*) requires the same measured quantity as reported in two or more papers; T3 (*heterogeneous multi-hop*) requires hops of genuinely different kinds; T4 (*three-retriever mandatory*) requires a concrete numeric or verbatim detail value plus two further hops of a different kind. Each question carries its gold answer, verbatim gold evidence spans, and the `arxiv_ids` of the contributing papers. A no-evidence genotype, whose correct answer is a statement that the corpus contains nothing, is generated but excluded from the reported set: its scoring depends on the judge accepting a negative answer, which the ragas correctness metric does not reliably reward.

D Additional Limitations

Limitations. Resolution of the smaller benchmark. `AI4EIC2023` has $n = 50$ questions and a per-question correctness difference with standard deviation 0.17, so it resolves gaps above roughly 10 points and nothing we compare is that far apart (Section B.1). Its rows are reported for completeness and for the contrast they draw with `LHCpubs-100QA`; no conclusion in this paper rests on a gap measured only there.

Baseline and retrieval-fusion controls. The single-call baseline and ModeRAG are our own implementations and share one harness, while the external systems are run as released (Section 4.1); the internal tool-set ablation is therefore the controlled comparison. In addition, the All Modes configuration runs a dense-vector pass for each planned graph task, so its comparison with Naive Only combines graph retrieval with additional vector retrieval calls. Future controlled comparisons should match retrieval-call budget and generator-visible context size.

Domain and benchmark scope. Both benchmarks cover high-energy-physics literature and use one LLM backend (`deepseek-v4-flash-ioa`) and one embedding model (BAAI/bge-m3); the ragas judges share the answer-generation backend. We chose `LHCpubs-100QA` to annotate cross-paper retrieval demands specific to the study and `AI4EIC2023` for an in-domain, mostly single-paper contrast. QASPER (Dasigi et al., 2021) targets information-seeking questions over NLP papers, MultiHop-RAG (Tang & Yang, 2024) uses a news corpus, and MuSiQue (Trivedi et al., 2022) tests compositional multi-hop reasoning; these benchmarks address useful but different domains and task constructions. We do not treat them as easy or unsuitable. Evaluating on them would test transfer beyond the present high-energy-physics setting, which remains future work. The shared judge backend may also correlate evaluation errors with generation errors, most visibly in the non-committal-answer rule that zeroes 18% of relevancy scores on `LHCpubs-100QA`.

External baselines. Table 5 pairs our Vanilla RAG and ModeRAG implementations with the released Microsoft GraphRAG, LightRAG and Agentic Hybrid RAG systems on the same questions, so it compares complete systems rather than isolating retrieval design. It does not exhaust the space of graph-retrieval systems, and the internal tool-set ablation remains the controlled comparison for the effect of the retrieval modes themselves.

Planner reliability. Decomposition and mode assignment are prompted, not trained, and under-decomposition is the dominant failure mode we observe. A router that detects when a single retrieval suffices would address both that failure and the cost in Section B.3; we do not evaluate one here.