DeepSieve: Information Sieving via LLM-as-a-Knowledge-Router

Anonymous ACL submission

Abstract

Large Language Models (LLMs) excel at many reasoning tasks but struggle with knowledge-intensive queries due to their inability to dynamically access up-to-date or domain-specific information. Retrieval-Augmented Generation (RAG) has emerged as a promising solution, enabling LLMs to ground their responses in external sources. However, existing RAG methods lack fine-grained control over both the query and source sides, resulting in noisy retrieval, shallow reasoning, and limited adaptability to heterogeneous knowledge sources. In this work, we introduce ***DeepSieve***, a novel RAG method that incorporates information sieving via LLM-as-a-knowledge-router. DeepSieve breaks down complex queries into structured sub-queries and recursively routes each to the most appropriate knowledge source, filtering out irrelevant information through a multi-stage information sieving process. This modular and transparent approach ensures that DeepSieve remains adaptable across diverse information needs. Experiments on three multi-hop QA benchmarks involving heterogeneous sources show that DeepSieve achieves greater reasoning depth, retrieval precision, and interpretability compared to conventional RAG approaches. Our codes are available at https://anonymous.4open.science/r/DeepSieve.

1 Introduction

Large language models (LLMs) have achieved remarkable progress across a wide range of natural language tasks and have demonstrated strong reasoning abilities, including in domains such as mathematics (Guo et al., 2025), fake news detection (Yi et al., 2025), and commonsense reasoning (Yao et al., 2023). Yet these LLMs often falter when facing knowledge-intensive questions that require up-to-date or domain-specific information (Guu et al., 2020; Asai et al., 2023; Fan et al., 2024). This limitation stems from the fixed nature of LLM parameters, which prevents dynamic access to external knowledge and frequently results in hallucinations or factually inaccurate outputs (Huang et al., 2025). Retrieval-Augmented Generation (RAG) has emerged as a powerful paradigm for equipping LLMs with access to external knowledge (Lewis et al., 2020; He et al.; Su et al., 2025), enabling them to tackle complex information-seeking tasks more effectively. Recent advances such as GraphRAG, HippoRAG, and so on (Gutierrez et al., 2024; Edge et al., 2024; Zhang et al., 2025) demonstrate the promise of structured and memory-augmented retrieval in improving factual accuracy and multi-hop reasoning.

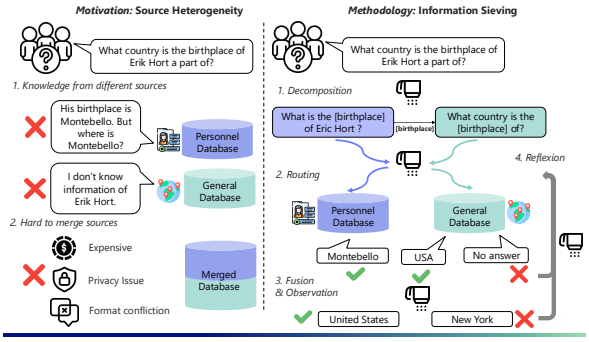

Figure 1: **Motivation and Overview:** Left: Compositional queries are hard to answer under source heterogeneity (e.g., structured, private, and unmergeable databases). Right: DeepSieve performs decomposition, source-aware routing, and iterative fusion to enable structured reasoning.

However, existing RAG systems still struggle with a fundamental limitation: the lack of an effective *information sieving* mechanism across the reasoning pipeline. In practice, this deficiency manifests in two critical forms: *query-side sieving* and *source-side sieving*. On the query side, most systems treat user queries as atomic units and directly retrieve information without decomposing or analyzing their underlying semantic structure (Trivedi et al., 2023; Yao et al., 2023; Lin et al., 2025). This

prevents the model from isolating key subgoals or reasoning steps, which is essential for multi-hop or compositional question answering.

On the source side, knowledge can be heterogeneous both across different sources (e.g., unstructured corpora, structured APIs, private databases) and within a single source that contains multiple domains or topics. However, most existing RAG systems retrieve from a flat, unified index without considering differences in domain, format, access modality, or granularity (Chen et al., 2017; Izacard and Grave, 2021; Gao et al., 2023). This often results in mismatched content, irrelevant retrievals, and unnecessary computational cost. Moreover, many knowledge sources cannot be merged into a single retrieval index because of privacy constraints, structural incompatibility, or deployment considerations (Mavi et al., 2023; Xu et al., 2025), further highlighting the need for selective and modular retrieval strategies.

To bridge this knowledge gap, we propose ***DeepSieve***, a framework that performs retrieval-augmented reasoning through multi-stage information sieving. As shown in the Figure 1, DeepSieve consists of four key components: *question decomposition*, *thought generation*, *data-source routing*, and *recursive reflexion*. The pipeline begins by decomposing a complex input query into a set of structured sub-questions using an LLM-based planner (Lin et al., 2025; Trivedi et al., 2022). Each sub-question is then independently processed to generate a "thought", a latent reasoning step that guides retrieval planning (Yao et al., 2023; Xu et al., 2023). Based on this thought, DeepSieve identifies the most appropriate source (e.g., API, SQL, RAG corpus) and generates a corresponding action plan. This source-aware routing builds on the idea of using LLMs as controllers for modular tool access (Schick et al., 2024; Xu et al., 2025).

Once evidence is retrieved, the system evaluates whether the information sufficiently addresses the sub-question. If the answer is invalid, DeepSieve enters a reflection loop to reassess the thought, revise the action plan, or reselect the source, iterating until resolution or timeout (Madaan et al., 2023; Baek et al., 2025). Finally, after all sub-questions are addressed, DeepSieve aggregates the answers into a coherent output using LLM module, which also generates intermediate rationales (Wei et al., 2022; Creswell and Shanahan, 2022).

This iterative, modular design allows DeepSieve to sieve and integrate knowledge progressively, making it particularly suited for multi-hop reasoning and heterogeneous information access. Empirically, DeepSieve consistently outperforms the baselines across all three benchmarks, achieving average F1/EM scores of 58.9/49.3 with DeepSeek-V3 and 51.2/41.4 with GPT-4o, surpassing both RAG (e.g., HippoRAG (Gutierrez et al., 2024)) and agentic (e.g., ReAct (Yao et al., 2023)) baselines, while using significantly fewer tokens. To sum up, our paper makes the following key contributions:

- We identify the structural and semantic heterogeneity of real-world knowledge sources as a core challenge in RAG. To address this, we propose the **DeepSieve** framework, which introduces *information sieving* and, for the first time, uses an **LLM-as-a-knowledge-router** to dynamically decompose queries and dispatch sub-questions to heterogeneous sources.

- Our experiments demonstrate that DeepSieve not only satisfies multi-source settings but also improves performance in standard single-source scenarios. Even when operating over a unified corpus, our system yields better retrieval precision and answer accuracy.

- We design DeepSieve as a modular and extensible method. It supports plug-and-play integration with diverse tools, retrieval backends, and RAG models. It provides a flexible backbone for future RAG architectures.

2 Methodology

We propose **DeepSieve**, a novel RAG method that performs information sieving across both query and source spaces. DeepSieve addresses the challenge of reasoning over heterogeneous knowledge sources, spanning diverse formats, access modalities, and domains, by employing a layered process of decomposition, routing, reflection, and fusion.

As shown in Figure 1 and Algorithm 1, DeepSieve decomposes input queries into subqueries, routes each to an appropriate *(Tool, Corpus)* pair, iteratively refines insufficient answers via reflexion, and ultimately fuses resolved subanswers into a coherent final response.

Notation. Let Q denote the original input query; let $\mathcal{S} = \{(T_k, C_k)\}$ denote the source set, where T_k is a tool and C_k is a corpus. DeepSieve performs the following modular operations:

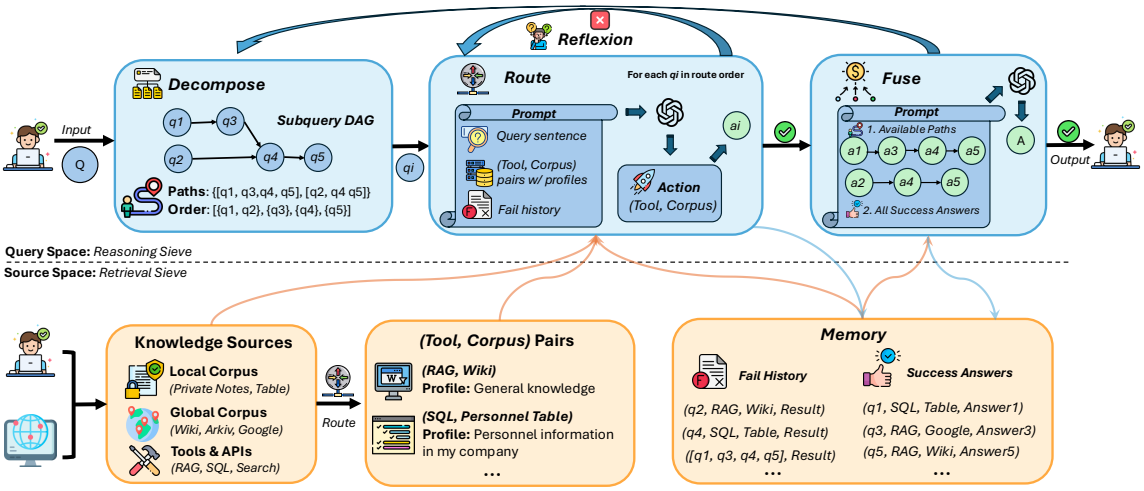

Figure 2: ***DeepSieve workflow*** with multi-step reasoning: A complex query is first decomposed into diverse subqueries, like *a directed acyclic graph (DAG)*. For each path in the DAG, an LLM generates a plan to select knowledge sources via routing. Failed retrievals will trigger re-routing or re-decomposition in the workflow. Retrieved subanswers are stored in memory and later fused across paths to form a final answer.

$\texttt{Decompose}(Q) \rightarrow \{q_i\}_{i=1}^{n}$: Decomposes the query into structured subqueries.
$\texttt{Route}(q_i, \mathcal{S}) \rightarrow s_i$: Selects a knowledge source $s_i = (T_i, C_i)$ from the source set $\mathcal{S}$.
$\texttt{Retrieve}(q_i, s_i) \rightarrow a_i$: Retrieves an answer a_i from source s_i for subquery q_i.
$\texttt{Reflect}(q_i, a_i) \rightarrow IF_REPLAN$: If a_i is insufficient, determines whether to replan q_i.
$\texttt{Fuse}(\{a_i\}) \rightarrow \hat{A}$: Aggregates all valid subanswers into a final response.

System Input and Execution. Given a query Q and a source pool $\mathcal{S} = \{(T_k, C_k)\}$ of tool–corpus pairs, DeepSieve process the query in 4 stages as shown in Algorithm 1 and illustrated from Section 2.1 to Section 2.4.

Algorithm 1: DeepSieve Pipeline

Input: Natural language query Q; source set $\mathcal{S}$
Output: Final answer $\hat{A}$

$\{q_1, \ldots, q_n\} \leftarrow \texttt{Decompose}(Q)$
foreach q_i *in execution order* **do**
  $(T_i, C_i) \leftarrow \texttt{Route}(q_i, \mathcal{S})$
  $a_i \leftarrow \texttt{Retrieve}(q_i, T_i, C_i)$
  $\texttt{IF_REPLAN} \leftarrow \texttt{Reflect}(q_i, a_i)$;
  if $\texttt{IF_REPLAN}$ **then**
    $(T_i, C_i) \leftarrow \texttt{Route}(q_i, \mathcal{S} \backslash \{(T_i, C_i)\})$;
    $a_i \leftarrow \texttt{Retrieve}(q_i, T_i, C_i)$;
  Store $(q_i, (T_i, C_i), a_i)$ into memory $\mathcal{M}$;
$\hat{A} \leftarrow \texttt{Fuse}(\{a_i\}_{i=1}^{n}, \mathcal{M})$ **return** $\hat{A}$

2.1 Stage I: Query Decomposition

Given a complex query $\mathcal{Q}$, DeepSieve first invokes an LLM-based planner to decompose it into a set of structured subquestions $\{q_1, q_2, \ldots, q_n\}$. This step acts as a *semantic sieve*, transforming monolithic input into a directed acyclic graph (DAG) of subgoals. Each node in the DAG represents an atomic reasoning unit, while edges capture dependency constraints resolved at fusion time.

2.2 Stage II: Knowledge Routing via LLM-as-Router

For each subquestion q_i, the system employs an LLM-based router to select a tool–corpus pair (T_i, C_i) from the source pool $\mathcal{S} = \{(T_k, C_k)\}$. This selection is guided by a structured routing prompt that encodes (i) the semantics of q_i, (ii) the profile metadata of each source (e.g., domain, format, privacy level), and (iii) a fail history $\mathcal{M}_{\text{fail}}$ of previous retrieval attempts.

The router returns a source $s_i = (T_i, C_i)$, and the system invokes the retrieval tool T_i on corpus C_i to obtain an answer candidate a_i.

2.3 Stage III: Observation and Reflexion

If the retrieved answer a_i is deemed unsatisfactory, e.g., incomplete, irrelevant, or ambiguous, DeepSieve triggers a reflexion step to reassess the current subquery q_i. Instead of modifying the subquery content, the system re-routes q_i to select an

alternative tool–corpus pair $s'_i = (T'_i, C'_i)$, and attempts retrieval again.

This process is guided by the memory module $\mathcal{M}$, which records all attempted subqueries and outcomes. Specifically, failed retrievals are stored in $\mathcal{M}_{\text{fail}}$ as tuples $(q_i, s_i, \texttt{Result})$, helping the router avoid redundant sources in future attempts. Successful results are stored in $\mathcal{M}_{\text{succ}}$ as (q_i, s_i, a_i), forming trusted evidence for final answer fusion.

2.4 Stage IV: Answer Fusion

After all subquestions have been individually resolved, DeepSieve invokes a fusion module to perform final aggregation. This module collects the set of successful subanswers $\{a_i\}_{i=1}^{n}$ from memory $\mathcal{M}_{\text{succ}}$, and synthesizes them into a globally coherent answer $\hat{A}$.

The fusion process leverages the DAG structure defined during query decomposition, which encodes both the reasoning order and dependency relationships among subquestions. It considers all valid reasoning paths that traverse the subquestion graph and selects consistent subanswers along these paths for inclusion. In cases where conflicting evidence is encountered, DeepSieve optionally performs global inference using an LLM to resolve contradictions and generate a unified response.

2.5 Modularity and Extensibility

DeepSieve is designed with a modular architecture that supports seamless integration of heterogeneous tools and knowledge sources. Each core component, decomposition, routing, retrieval, reflexion, and fusion, can be independently replaced or extended without modifying the overall control flow. Knowledge sources are abstracted as `(Tool, Corpus)` pairs, each annotated with a natural language profile that guides source selection during routing. This abstraction enables plug-and-play extension: adding a new retriever (e.g., `BM25`, `FAISS`, `ColBERTv2`) or a new source (e.g., SQL, API) only requires registering its wrapper and profile. The system also scales naturally to multi-source settings through semantic clustering or source-specific wrappers, eliminating the need for index merging or schema unification.

3 Experiments

We evaluate DeepSieve on multi-hop QA benchmarks to answer four core research questions:

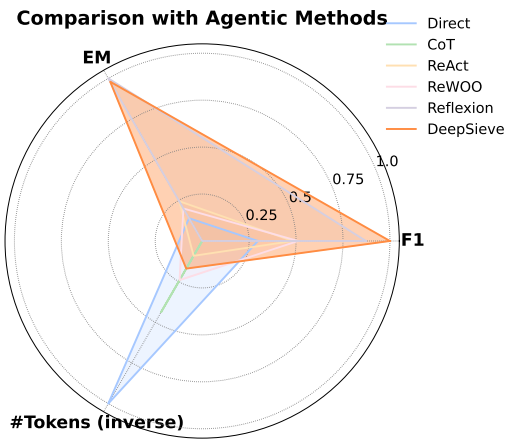

Figure 3: Normalized radar plot comparing agentic methods based on their average scores across all benchmarks. The plot evaluates methods across three dimensions: F1 score, EM score, and token efficiency represented as #Tokens (inverse). All metrics are normalized, and a higher value indicates better performance for each axis. A larger enclosed area signifies a superior trade-off between accuracy and computational cost.

- **RQ1:** Does DeepSieve outperform traditional RAG baselines?
- **RQ2:** Is DeepSieve more efficient in inference cost than other agentic RAG methods?
- **RQ3:** How do decomposition, routing, and reflexion contribute to overall performances, respectively?
- **RQ4:** Can DeepSieve adapt flexibly across different retrievers and modular knowledge source configurations?

3.1 Experimental Setup

We evaluate on 3 benchmarks: MuSiQue (Trivedi et al., 2022), 2WikiMultiHopQA (Fu et al., 2021), and HotpotQA (Yang et al., 2018), following IR-CoT (Trivedi et al., 2023) to construct retrieval corpora with both supporting and distractor passages (1,000 dev questions). We use DeepSeek-V3 (Guo et al., 2025) and GPT-4o (OpenAI et al., 2024) as the backbone LLM. To simulate source heterogeneity, we partition each dataset into *local* and *global* segments using LLM-based profiles (see Appendix G). DeepSieve performs subquestion-level routing over these two source modules, allowing modular reasoning across simulated access boundaries. Since existing baselines do not support multi-source retrieval or dynamic source selection, we evaluate them using the original corpus, which is

the same as the combination of local and global segments mentioned above, to ensure fair comparison. This setup allows us to isolate the impact of modular routing while maintaining compatibility across methods.

Baselines. We compare against IRCoT (Trivedi et al., 2023), ColBERTv2 (Santhanam et al., 2022), HippoRAG (Gutierrez et al., 2024), and RAPTOR (Sarthi et al., 2024) as representative RAG paradigms baseline. We also include ReAct (Yao et al., 2023), ReWOO (Xu et al., 2023), Reflexion (Madaan et al., 2023), and Chain-of-Thought (CoT) (Wei et al., 2022) as reasoning and agentic baselines because DeepSieve utilizes not only RAG algorithm. The details of these baselines and their setups are listed in Appendix F.

Metrics. We report Exact Match (EM) and F1 scores to evaluate answer correctness, where EM measures exact string match and F1 accounts for token-level overlap. To assess inference cost, we track the total number of tokens generated by the LLM across all reasoning steps.

3.2 Main Performance Comparison (RQ1, RQ4)

In this section, we aim to evaluate whether DeepSieve improves answer accuracy across multiple multi-hop QA datasets, and how it compares with both pure RAG and reasoning-based baselines. Table 1 includes two DeepSieve variants to demonstrate its adaptability across retrieval paradigms: **Naive RAG**, which retrieves from a flat corpus using all-MiniLM-L6-v2, and **GraphRAG**, which builds on structure-aware retrieval via document-level links following the GraphRAG setting. Both variants use DeepSeek-V3 and GPT-4o as the backbone LLM with the identical decoding parameters, which illustrates DeepSieve's modularity across different retrieval methods.

DeepSieve(Naive RAG) achieves the best F1 score on MuSiQue (46.8, +13.5 over IRCoT+HippoRAG) and 2WikiMultiHopQA (68.4, +5.3), highlighting the benefit of structured decomposition and source-aware retrieval. On HotpotQA, DeepSieve performs below RAPTOR, which benefits from the design of HotpotQA itself that favors models with entity linking and graph construction. However, unlike RAPTOR, DeepSieve performs better on average and operates in a fully online and modular manner, without requiring any static graph preprocessing or clustering. On average, DeepSieve (Naive RAG) achieves an F1 of 58.9, outperforming all baselines.

Under the GPT-4o setting, DeepSieve (Naive RAG) achieves an F1 of 61.7 on HotpotQA (Table 2), outperforming other multi-hop reasoning frameworks such as Chain-of-Thought (30.8), ReAct (39.6), and ReWOO (40.1). It also approaches the performance of Reflexion (46.7 vs. 49.3), while employing a more modular design with explicit decomposition and source-specific coordination.

Takeaway: RQ1

DeepSieve achieves the best average F1 across all datasets, outperforming both pure RAG baselines and agentic RAG method baselines without relying on static graphs.

3.3 Efficiency Comparison with Reasoning and Agentic Methods (RQ1, RQ2)

We then evaluate whether DeepSieve achieves its performance gains efficiently by comparing LLM token usage with other LLM-based reasoning and agentic methods. As shown in Table 2, DeepSieve attains higher accuracy while using significantly fewer tokens. On HotpotQA, DeepSieve achieves the highest F1 score (49.0) and EM (61.6), with an average of only 3.9K tokens per query, compared to Reflexion (37.9K) and ReAct (9.8K).

To better illustrate the performance–efficiency trade-off, Figure 3 shows a normalized radar plot across three dimensions: F1, EM, and inverse token usage. DeepSieve covers the largest area, demonstrating strong overall performance across all metrics. While ReAct and Reflexion achieve similar accuracy, they require far more tokens. In contrast, Direct and CoT are efficient but lag in accuracy. By balancing all three dimensions, DeepSieve stands out as a cost-effective LLM-based RAG system.

Takeaway: RQ2

DeepSieve uses significantly fewer tokens than other LLM-based systems while achieving higher accuracy, showing strong cost-effectiveness.

3.4 Ablation Study (RQ3)

To understand the contribution of each module within DeepSieve, we conduct an ablation study to assess its individual and combined effects on system performance. Our results indicate that re-

Table 1: Exact Match (EM) and F1 scores on MuSiQue, 2WikiMultihopQA, and HotpotQA under DeepSeek-V3 and GPT-4o. DeepSieve is evaluated using a simulated heterogeneous setup with local/global source partitioning; baselines use the merged corpus. Bold: best; ↑: improvement over best non-DeepSieve; superscripts indicate F1/EM gains.

	Retriever	MuSiQue		2Wiki		HotpotQA		Average	
		EM	F1	EM	F1	EM	F1	EM	F1
DeepSeek-V3	Naive RAG (MiniLM)	20.5	26.1	26.4	31.3	30.4	42.8	25.8	33.4
	ColBERTv2	17.1	27.2	32.9	43.8	42.9	57.2	31.0	42.7
	HippoRAG	19.4	27.9	44.9	58.9	40.8	55.3	35.0	47.4
	IRCoT + HippoRAG	21.4	33.4	46.5	63.1	45.1	58.9	37.7	51.8
	GraphRAG	7.7	9.7	30.1	33.2	45.9	55.2	27.9	32.7
	RAPTOR	18.2	28.9	38.6	52.1	52.9	66.5	37.5	50.2
	DeepSieve (Naive RAG)	**36.0**$^{+14.6\uparrow}$	**46.8**$^{+13.4\uparrow}$	**62.8**$^{+16.3\uparrow}$	**68.4**$^{+5.3\uparrow}$	49.0	61.6	**49.3**$^{+11.6\uparrow}$	**58.9**$^{+7.1\uparrow}$
	DeepSieve (GraphRAG)	30.0$^{+8.6\uparrow}$	36.6$^{+3.2\uparrow}$	49.2$^{+2.7\uparrow}$	53.8	49.4	61.6	42.9$^{+5.2\uparrow}$	50.7
GPT-4o	Naive RAG (MiniLM)	19.4	27.1	26.7	29.8	29.8	44.5	25.3	33.8
	ColBERTv2	15.5	26.4	33.4	43.3	43.4	57.7	30.8	42.5
	HippoRAG	19.2	29.8	46.6	59.5	41.8	55.0	35.9	48.1
	IRCoT + HippoRAG	21.9	33.3	47.7	62.7	45.7	56.2	38.4	50.7
	GraphRAG	8.7	10.9	29.8	33.3	44.3	52.2	25.6	31.0
	RAPTOR	18.8	29.7	39.3	51.6	52.4	66.3	36.8	49.2
	DeepSieve (Naive RAG)	**26.7**$^{+4.8\uparrow}$	**36.6**$^{+3.3\uparrow}$	**48.3**$^{+0.6\uparrow}$	55.2	49.3	61.7	**41.4**$^{+3.0\uparrow}$	**51.2**$^{+0.5\uparrow}$
	DeepSieve (GraphRAG)	20.3	31.1	27.5	38.9	44.6	53.5	32.7	43.2

Table 2: Comparison of Reasoning & Agentic RAG paradigms on **HotpotQA** using GPT-4o in terms of F1, EM, and token usage.

Paradigm	F1	EM	#Tokens
Direct	36.2	28.0	**55.5**
CoT	30.8	22.4	481.9
ReAct	39.6	32.2	9795.1
ReWOO	40.1	30.4	1986.2
Reflexion	46.7	**62.5**	37893.0
DeepSieve	**49.3**	61.7	3926.6

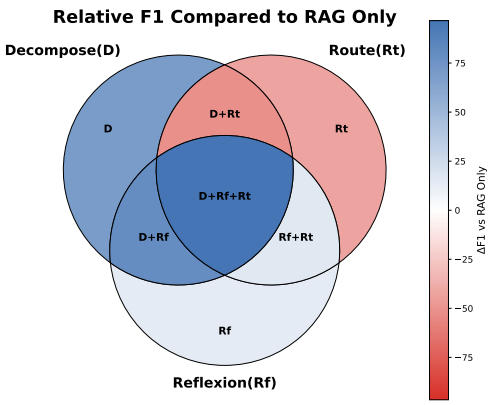

Figure 4: Ablation study of F1 score improvements (blue) and declines (red) over Naive RAG across datasets. Color intensity corresponds to the magnitude of performance change, with darker shades indicating stronger effects.

moving any module reduces performance, with **Reflexion** and **Decomposition** being most critical. Table 3 shows that disabling Reflexion drops F1 from 68.4 to 15.4 on 2WikiMultihopQA, and removing Decomposition results in a 17.5-point drop on MuSiQue, highlighting their central roles in multi-hop reasoning.

In contrast, using **Routing** alone performs poorly and even slightly reduces accuracy on HotpotQA. This is likely because HotpotQA contains fewer source ambiguities, so the added complexity of Routing does not provide enough benefit. However, Figure 4 shows that when Routing is combined with Decomposition and Reflexion (D+Rt+Rf), performance improves consistently across all datasets. While the D+Rf setup is already strong, adding Routing further boosts robustness and retrieval accuracy, especially in more diverse settings. Importantly, Routing is essential for DeepSieve's ability to handle multi-source, heterogeneous knowledge. The fact that Routing matches prior performance without sacrificing accuracy in these more general scenarios demonstrates its effectiveness. Even in datasets without pronounced source heterogeneity, it generalizes well.

Other than performance, Figure 5 details the token consumption of each module. The Decompo-

Table 3: Ablation study evaluating the performance contribution of each DeepSieve component. It shows how performance changes when components are removed ('w/o') or used in isolation. For each dataset, scores are presented in the format EM | F1 Score.

Setting	Decomp.	Routing	Reflect.	HotpotQA	2Wiki.	MuSiQue
Full DeepSieve	✓	✓	✓	49.0 | 61.6	**62.8 | 68.4**	**36.0 | 46.8**
w/o Reflexion	✓	✓		17.3 | 21.6	15.2 | 15.4	5.4 | 11.5
w/o Routing	✓		✓	**54.0 | 65.8**	61.4 | 64.7	34.5 | 44.6
w/o Decomposition		✓	✓	33.1 | 41.1	35.2 | 36.9	22.0 | 28.6
Decomposition Only	✓			52.3 | 62.1	60.3 | 63.1	33.5 | 44.0
Routing Only		✓		32.1 | 42.9	4.2 | 4.6	6.4 | 9.8
Reflexion Only			✓	33.2 | 43.0	28.7 | 33.2	21.2 | 28.1
Naive RAG				30.4 | 42.8	26.4 | 31.3	20.5 | 26.1

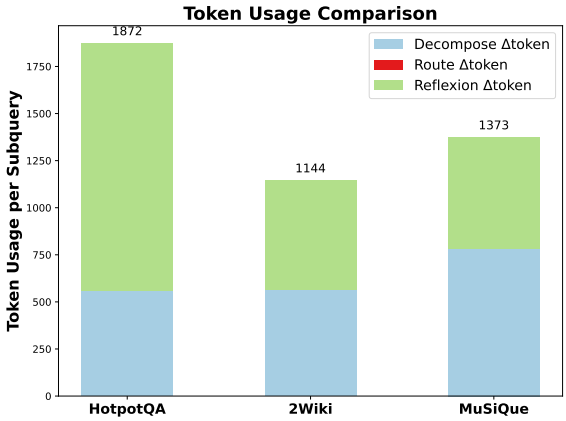

Figure 5: A comparison of token costs for each stage of the framework. The stacked bars illustrate the cumulative token usage per subquery, showing the base cost of Decomposition (blue), plus the additional costs from Routing (red) and Reflexion (green).

sition stage consistently establishes a significant baseline cost, ranging from approximately 550 to 780 tokens per query. In contrast, the additional overhead from the Routing stage is negligible across all datasets. The most substantial portion of the additional cost is contributed by the Reflexion stage. This cost analysis underscores the strategic design of our framework. With its minimal token footprint, the Routing module is the key component that enables DeepSieve to handle heterogeneous data sources and provides a net performance boost when combined with the other modules. Conversely, while Decomposition and Reflexion account for the majority of the token cost, the ablation results confirm they are indispensable for achieving high accuracy, as removing either leads to a dramatic drop in performance.

Finally, we evaluate the **Fusion** module's effectiveness by comparing fused outputs against final-subquery answers before fusion. Results demonstrate consistent accuracy improvements, with a high fix-to-error ratio confirming robust aggregation of routed subquery results as Figure 6 shows.

Takeaway: RQ3

Decomposition and reflexion are key to accuracy. Routing alone may underperform, but in combination, it improves robustness and handles heterogeneous knowledge sources without sacrificing performance.

3.5 Modular Design and Adaptability (RQ4)

DeepSieve supports both Naive RAG and GraphRAG retrieval setups, demonstrating its adaptability and modular design. To simulate heterogeneous corpora, we partition each dataset into *local* and *global* segments, enabling subquestion-level routing to different sources. DeepSieve achieves strong performance across both retrieval modes, outperforming prior RAG baselines while maintaining flexible source integration. Furthermore, to enable multi-format access, we implement modular interfaces in our framework(like SQL experiment results in Appendix I), supporting potential integration with databases and APIs.

Takeaway: RQ4

DeepSieve generalizes across retrievers and source configurations, with modular support for structured sources (e.g., SQL, JSON) implemented.

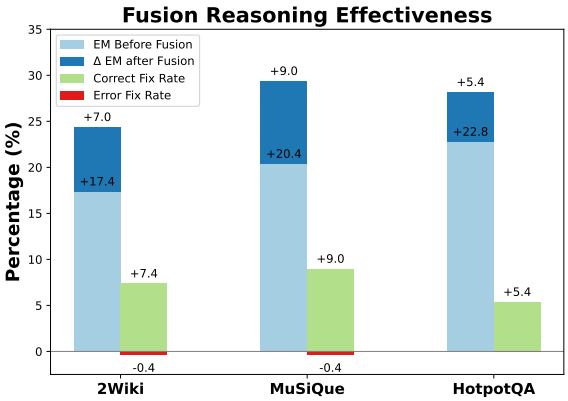

Figure 6: Effect of fusion reasoning on EM scores of DeepSieve(GraphRAG). For each dataset, the left bar shows EM before and after fusion (blue), and the right bar shows how often fusion fixes incorrect answers (green) or corrupts correct ones (red).

4 Related Work

We position our work at the intersection of four key research directions: decomposition for multi-hop reasoning, RAG with heterogeneous sources, LLM-based routing, and reflexion RAG method.

Multi-Hop Reasoning and Decomposition Multi-hop question answering (QA) requires breaking down complex queries into simpler subtasks. Decomposed Prompting proposes a modular planner–executor framework to tackle complex reasoning tasks (Khot et al., 2023). ADaPT dynamically determines when to decompose using planner-based feedback loops (Prasad et al., 2024). DISC improves scalability by dynamically decomposing inference steps with memory efficiency (Light et al., 2025). SealQA integrates decomposition and verification into search-augmented LMs (Cui et al., 2024). Ye et al. formalize decomposition as a representation-quality check in RAG (Zeng et al., 2025). These methods enhance reasoning depth and modular execution but do not address source- or tool-aware routing in heterogeneous environments.

RAG with Heterogeneous Sources RAG systems augment LLMs with both structured and unstructured knowledge (Lewis et al., 2020; Talmor et al., 2021). HippoRAG introduces memory alongside structured retrieval (Gutierrez et al., 2024). HippoRAG2 extends it with continual memory using clustering and profiling (Gutiérrez et al., 2025). InfuserKI enhances LLMs with knowledge graphs via infusing (Wang et al., 2024). AutoSchemaKG automates schema induction for knowledge graph construction from web corpora (Bai et al., 2025). These approaches handle heterogeneous memory and structure, but still rely on flat retrieval indexes without per-subquestion navigation.

LLM as Router for Source-Aware Retrieval Recent work explored by LLMs to control retrieval behavior, but often under homogeneous assumptions. Probing-RAG (Baek et al., 2025) leverages LLMs' self-reflexion to guide document selection, but operates over a single unified corpus. OmniRouter (Mei et al., 2025) introduces cost-aware retrieval routing over sub-indices, assuming similar retrieval formats. Toolformer (Schick et al., 2024) fine-tunes LLMs to call APIs, yet does not support structured routing or modular tool orchestration. By contrast, DeepSieve treats routing as a decision step, matching each subquestion to a specific *source profile* drawn from a heterogeneous set of corpora and tools (e.g., SQL, APIs, RAG corpora).

Agentic Methods Agentic systems empower LLMs to reason, plan, and act over multi-step inference chains. Some agentic methods are also used for RAG tasks. ReAct (Yao et al., 2023) merges tool use and thought generation in a unified loop. ReWOO (Xu et al., 2023) decouples retrieval from reasoning to reduce token cost. MA-RAG (Nguyen et al., 2025) introduces agentic collaboration via CoT-based subquerying. In contrast, DeepSieve builds upon this line of work by introducing an explicit modular planner-router-retriever loop, enabling fine-grained reasoning over decomposed subquestions, recursive reflexion over failures, and heterogeneous tool dispatch.

5 Conclusion

We present DeepSieve, a RAG method that addresses the limitations of traditional RAG pipelines in handling compositional queries and heterogeneous knowledge sources. DeepSieve decomposes queries into subquestions, routes each to the most suitable source using an LLM-based router, and performs retrieval and reflexion before aggregating final answers. Through extensive experiments on three multi-hop QA benchmarks, we demonstrate that DeepSieve achieves strong performance across different LLMs, outperforming both RAG baselines and agentic baselines. The method also generalizes across retrieval backends (Naive RAG vs. GraphRAG) and supports the interfaces for SQL and JSON sources in the source code.

Limitations

While DeepSieve demonstrates strong performance in modular reasoning and heterogeneous source routing, there remain two key directions for further enhancement: First, the current routing mechanism selects only a coarse-grained (tool, source) pair for each subquestion. This limits the system's ability to leverage fine-grained configurations, such as tool-specific parameters (e.g., retrieval depth, temperature, API mode) or function-level APIs. Future work could extend the action space to support parameterized tool selection, enabling more adaptive behavior and cost-aware decisions at inference time. Second, although DeepSieve supports modular integration of private sources, it treats all subquestions uniformly across users. In real-world settings, however, different users may have personalized knowledge graphs, access patterns, or preferences. A promising direction is to incorporate personalized routing and memory modules, allowing the LLM to learn user-specific retrieval paths, preferred sources, or task priors, thus enabling long-term adaptation and user-centric QA behavior. We believe these extensions will further enhance the controllability and personalization of RAG systems in both industrial and research applications.

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations (ICLR)*.

Ingeol Baek, Hwan Chang, ByeongJeong Kim, Jimin Lee, and Hwanhee Lee. 2025. Probing-RAG: Self-probing to guide language models in selective document retrieval. In *Findings of Proceedings of the 2025 Conference of the North628 American Chapter of the Association for Computa-629 tional Linguistics (NAACL)*, pages 3287–3304, Albuquerque, New Mexico. Association for Computational Linguistics.

Jiaxin Bai, Wei Fan, Qi Hu, Qing Zong, Chunyang Li, Hong Ting Tsang, Hongyu Luo, Yauwai Yim, Haoyu Huang, Xiao Zhou, Feng Qin, Tianshi Zheng, Xi Peng, Xin Yao, Huiwen Yang, Leijie Wu, Yi Ji, Gong Zhang, Renhai Chen, and Yangqiu Song. 2025. Autoschemakg: Autonomous knowledge graph construction through dynamic schema induction from web-scale corpora. *Preprint*, arXiv:2505.23628.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Antonia Creswell and Murray Shanahan. 2022. Selection-inference: Exploiting large language models for interpretable logical reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Leyang Cui, Yunan Xie, Qinyuan Liu, Dian Yu, Zhiwei Liu, Shuohui Shao, Mingzhe Sun, and Graham Neubig. 2024. Sealqa: Raising the bar for reasoning in search-augmented language models. *arXiv preprint arXiv:2506.01062*.

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*.

Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 6491–6501.

Ruiliu Fu, Han Wang, Xuejun Zhang, Jun Zhou, and Yonghong Yan. 2021. Decomposing Complex Questions Makes Multi-Hop QA Easier and More Interpretable. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 169–180, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202, pages 10764–10785. PMLR.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *DeepSeek-AI*.

Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Daniel Gutiérrez, Xueguang Dai, Xi Victoria Lin, Chunting Liu, and et al. 2025. Ragmemory v2: Non-parametric continual memory for language models. In *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented

language model pre-training. In *International conference on machine learning (ICML)*, pages 3929–3938. PMLR.

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. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and 1 others. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems (TIST)*, 43(2):1–55.

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

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2023. Decomposed prompting: A modular approach for solving complex tasks. *Preprint*, arXiv:2210.02406.

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. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS)*, pages 9459–9474.

Jonathan Light, Wei Cheng, Wu Yue, Masafumi Oyamada, Mengdi Wang, Santiago Paternain, and Haifeng Chen. 2025. Disc: Dynamic decomposition improves llm inference scaling. *Preprint*, arXiv:2502.16706.

Teng Lin, Yizhang Zhu, Yuyu Luo, and Nan Tang. 2025. SRAG: Structured retrieval-augmented generation for multi-entity question answering over wikipedia graph. arXiv preprint. *Preprint*, arXiv:2503.01346.

Aman Madaan, Bill Yuchen Lin, and et al. 2023. Self-refine: Iterative refinement with self-feedback. In *Neural Information Processing Systems (NeurIPS)*.

Vaibhav Mavi, Abulhair Saparov, and Chen Zhao. 2023. Retrieval-Augmented Chain-of-Thought in Semi-structured Domains. In *Proceedings of the Natural Legal Language Processing Workshop 2023*, pages 178–191, Singapore. Association for Computational Linguistics.

Kai Mei, Wujiang Xu, Shuhang Lin, and Yongfeng Zhang. 2025. Omnirouter: Budget and performance controllable multi-llm routing. *Preprint*, arXiv:2502.20576.

Thang Nguyen, Peter Chin, and Yu-Wing Tai. 2025. Marag: Multi-agent retrieval-augmented generation via collaborative chain-of-thought reasoning. *Preprint*, arXiv:2505.20096.

OpenAI, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, and 400 others. 2024. Gpt-4o system card. *Preprint*, arXiv:2410.21276.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. 2024. Adapt: As-needed decomposition and planning with language models. *Preprint*, arXiv:2311.05772.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. ColBERTv2: Effective and efficient retrieval via lightweight late interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 3715–3734, Seattle, United States. Association for Computational Linguistics.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. 2024. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *The 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Timo Schick, Tilman Dwivedi-Yu, Hinrich Schütze, and et al. 2024. Toolformer: Language models can teach themselves to use tools. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*.

Weihang Su, Yichen Tang, Qingyao Ai, Junxi Yan, Changyue Wang, Hongning Wang, Ziyi Ye, Yujia Zhou, and Yiqun Liu. 2025. Parametric retrieval augmented generation. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval(SIGIR)*, SIGIR '25, page 1240–1250, New York, NY, USA. Association for Computing Machinery.

Alon Talmor, Ori Yoran, Amnon Catav, Dan Lahav, Yizhong Wang, Akari Asai, Gabriel Ilharco, Hannaneh Hajishirzi, and Jonathan Berant. 2021. Multimodal{qa}: complex question answering over text, tables and images. In *International Conference on Learning Representations*.

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 (TACL)*, 10:539–554.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving Retrieval with Chain-of-Thought Reasoning for Knowledge-Intensive Multi-Step Questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics(ACL)*, pages 10014–10037, Toronto, Canada. Association for Computational Linguistics.

Fali Wang, Runxue Bao, Suhang Wang, Wenchao Yu, Yanchi Liu, Wei Cheng, and Haifeng Chen. 2024. Infuserki: Enhancing large language models with knowledge graphs via infuser-guided knowledge integration. In *Findings of the Association for Computational Linguistics (EMNLP Findings)*, pages 3675–3688.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv preprint. *Preprint*, arXiv:2201.11903.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, and Dongkuan Xu. 2023. ReWOO: Decoupling reasoning from observations for augmented language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Hao Xu, Bowen Zhang, Minghao Guo, Yuchen Xie, and Xiang Chen. 2025. Amem: Agentic memory for llms via modular retrieval and source-aware routing. In *ACL 2025 (under review)*.

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(EMNLP)*.

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

Jingyuan Yi, Zeqiu Xu, Tianyi Huang, and Peiyang Yu. 2025. Challenges and innovations in llm-powered fake news detection: A synthesis of approaches and future directions. *Preprint*, arXiv:2502.00339.

Shenglai Zeng, Jiankun Zhang, Bingheng Li, Yuping Lin, Tianqi Zheng, Dante Everaert, Hanqing Lu, Hui Liu, Hui Liu, Yue Xing, Monica Xiao Cheng, and Jiliang Tang. 2025. Towards knowledge checking in retrieval-augmented generation: A representation perspective. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics(ACL)*, pages 2952–2969, Albuquerque, New Mexico. Association for Computational Linguistics.

Qinggang Zhang, Shengyuan Chen, Yuanchen Bei, Zheng Yuan, Huachi Zhou, Zijin Hong, Junnan Dong, Hao Chen, Yi Chang, and Xiao Huang. 2025. A survey of graph retrieval-augmented generation for customized large language models. *arXiv preprint arXiv:2501.13958*.

A Case Study: Error Avoidance via Decomposition and Routing

To demonstrate the benefits of our routing-based, multi-stage RAG pipeline, we present several qualitative examples in which DeepSieve significantly outperforms a standard flat RAG baseline. While quantitative metrics provide a global summary, these case studies are crucial for illustrating the distinct types of reasoning and retrieval failures that arise in open-domain multi-hop QA—and how DeepSieve's modular design mitigates them.

Each case highlights a representative failure mode of Pure RAG, such as hallucinating unsupported facts, making entity confusions due to semantic proximity, failing to perform multi-step inference, or being brittle to partial retrieval. In contrast, DeepSieve breaks down complex queries into structured subqueries, dynamically routes each subquery to the most relevant knowledge source, and employs reflexion to recover when intermediate reasoning fails.

The selected queries capture diverse challenges commonly encountered in real-world QA tasks:

- **Resolution of nested references** (Case 1): Answering requires tracing multiple entity links, e.g., identifying a person via another person's biography.
- **Interpretation of entity-to-entity relationships** (Case 2): Understanding geographic containment and relational structure between entities.
- **Robustness to early-stage retrieval errors** (Case 3): Overcoming noisy or misleading top-k passages by revising and retrying.
- **Disambiguation in entity-rich settings** (Case 4): Navigating multiple overlapping or semantically similar names across corpora.
- **Temporal linking across events** (Case 5): Connecting causally or chronologically related events that span different documents.

In all cases, we contrast DeepSieve with a Pure RAG system, which retrieves directly from a merged corpus without decomposition, source-specific routing, or any fallback mechanisms. Such flat pipelines often retrieve semantically similar but irrelevant contexts or hallucinate answers in the absence of explicit grounding.

Case 1: Decomposition Avoids Hallucination

Query: Who is the husband of the woman who founded the Flying Doctors service in Nigeria?

Pure RAG (Failure)

Top-k Retrieved:

- "Flying Doctors Nigeria was established to provide air ambulance services across West Africa."
- "History of emergency medical care in Nigeria."
- "Private medical companies in Africa."

Answer: Dr. Oluyemi
Error: No founder info; hallucinated name.

In contrast, DeepSieve decomposes the query into reasoning steps and retrieves targeted evidence from relevant sources.

DeepSieve (Success)

Decomposition:

- Q1: Who founded Flying Doctors? → **Dr. Ola Orekunrin**
- Q2: Who is her husband? → No public info

Routing: Q1 → Global, Q2 → Local
Top-k Retrieved:

- "Dr. Ola Orekunrin founded the Flying Doctors Nigeria service."
- "Medical entrepreneurship in West Africa."

Final Answer: No public record of her husband.

In this case, Pure RAG fails because the question contains a nested structure that requires identifying the founder before inferring her marital status. Without decomposition, the model hallucinates a name (Dr. Oluyemi) not supported by retrieved content. DeepSieve correctly separates the query into steps and prevents hallucination by explicitly handling reference resolution.

Case 2: Routing Improves Corpus Precision

Query: What country is the birthplace of Erik Hort a part of?

Pure RAG (Failure)

Top-k Retrieved:

- "Erik Hort was born in Montebello."
- "Cinderella attended the royal ball."

Answer: United States
Error: Guesswork; no reasoning trace.

By decomposing the query and rerouting subquestions, DeepSieve uncovers geographical dependencies that Pure RAG missed.

DeepSieve (Success)

Decomposition:

- Q1: Who was born in Montebello? → Erik Hort
- Q2: What state is Montebello in? → New York
- Q3: What country is New York in? → United States

Routing: Local → Global → Global
Top-k Retrieved:

- "Montebello is located in New York."
- "New York is one of the 50 US states."

Final Answer: United States

Here, a seemingly straightforward question actually requires multiple geographic hops. The Pure RAG system answers correctly by chance but lacks any justification. DeepSieve surfaces the latent structure, allowing the answer to be transparently derived through chain-of-thought and cross-source coordination.

Case 3: Reflexion Corrects Early Failure

Query: What is one of the stars of "The Newcomers" known for?

Pure RAG (Failure)

Top-k Retrieved:

- "Chris Evans starred in 'The Newcomers'."
- "Kate Bosworth acted in several teen films."

Answer: Dano is an indie film actor.
Error: Off-target guess; no grounding.

DeepSieve's reflexion mechanism re-queries when early steps fail, correcting misaligned or unsupported answers.

DeepSieve (Success via Reflexion)

Decomposition:

- Q1: Who starred in "The Newcomers"? → Chris Evans
- Q2: What is Chris Evans known for? → Captain America

Routing: Local → Global
Top-k Retrieved:

- "Chris Evans played Captain America in the Marvel series."

Final Answer: Captain America

Reflexion plays a key role when intermediate retrieval fails. Here, the initial step is successful, but the answer to the second query is incorrect. DeepSieve detects this and re-attempts the step with an updated subquery. The correct answer emerges only after this re-routing, which is not possible in flat RAG systems.

Case 4: Routing Prevents Misleading Associations

Query: Which Harry Potter character was played by Robbie Coltrane?

Pure RAG (Failure)

Top-k Retrieved:

- "Robbie Coltrane is a British actor."
- "He appeared in several comedies."

Answer: Newt Scamander
Error: Wrong movie, wrong actor.

Naive RAG suffers from semantically similar distractions. DeepSieve routes to the most accurate corpus for disambiguation.

DeepSieve (Success)

Decomposition:

- Q1: Who is Robbie Coltrane? → Actor in Harry Potter
- Q2: What character did he play? → Rubeus Hagrid

Routing: Global → Global
Top-k Retrieved:

- "Robbie Coltrane portrayed Hagrid in all eight Harry Potter films."

Final Answer: Rubeus Hagrid

This example shows how semantic similarity can be misleading. The name "Robbie Coltrane" triggers distractor documents about unrelated roles. DeepSieve's global routing ensures selection of contextually relevant facts, avoiding confusion with similarly structured but irrelevant entries.

Case 5: Decomposition Enables Knowledge Linking

Query: Who succeeded the Prime Minister that resigned during the Brexit vote?

Pure RAG (Failure)

Top-k Retrieved:

- "Brexit led to political unrest in the UK."

Answer: Boris Johnson
Error: Lucky guess; no traceable chain.

Chained subquestions let DeepSieve explicitly link political timelines, uncovering correct facts.

DeepSieve (Success)

Decomposition:

- Q1: Who was UK PM during Brexit vote? → David Cameron
- Q2: Who succeeded David Cameron? → Theresa May

Routing: Global → Global
Top-k Retrieved:

- "David Cameron resigned after Brexit."
- "Theresa May beat Cameron as PM."

Final Answer: Theresa May

This final case exemplifies how multi-hop question answering often entails chaining temporally or causally linked events. Without decomposition, the Pure RAG baseline simply guesses the answer based on limited evidence. DeepSieve reconstructs the chain of political succession through its decomposed subqueries, leading to a verifiable and accurate response.

Case 6: Decomposition Avoids Hallucination

Query: Who is the husband of the woman who founded the Flying Doctors service in Nigeria?

Pure RAG (Failure)

Top-k Retrieved:

- "Flying Doctors Nigeria was established to provide air ambulance services across West Africa."
- "History of emergency medical care in Nigeria."
- "Private medical companies in Africa."

Answer: Dr. Oluyemi
Error: No founder info; hallucinated name.

In contrast, DeepSieve decomposes the query into reasoning steps and retrieves targeted evidence from relevant sources.

DeepSieve (Success)

Decomposition:

- Q1: Who founded Flying Doctors? → **Dr. Ola Orekunrin**
- Q2: Who is her husband? → No public info

Routing: Q1 → Global, Q2 → Local
Top-k Retrieved:

- "Dr. Ola Orekunrin founded the Flying Doctors Nigeria service."
- "Medical entrepreneurship in West Africa."

Final Answer: No public record of her husband.

In this case, Pure RAG fails because the question contains a nested structure that requires identifying the founder before inferring her marital status. Without decomposition, the model hallucinates a name (Dr. Oluyemi) not supported by retrieved content. DeepSieve correctly separates the query into steps and prevents hallucination by explicitly handling reference resolution.

Case 7: Multi-Source Reasoning with SQL and RAG

Query: Which scientist born in the 19th century is known for discovering radioactivity, and what country did she live in during World War I?

Pure RAG (Failure)

Top-k Retrieved:

- "Radioactivity was discovered in the late 19th century."
- "Famous scientists in chemistry and physics."
- "Women in science and war."

Answer: Lise Meitner
Error: Missed date constraint; incorrect person selected.

In contrast, DeepSieve uses SQL to filter candidates by birth year, then RAG to confirm contributions and residency.

DeepSieve (Success)

Decomposition:

- Q1: Which scientists were born in the 19th century? → [SQL: Marie Curie, others]
- Q2: Who among them discovered radioactivity? → **Marie Curie**
- Q3: Where did she live during World War I? → France

Routing: Q1 → SQL, Q2/Q3 → Global RAG
Top-k Retrieved:

- "Marie Curie was born in 1867 and discovered radioactivity."
- "She lived in Paris and worked in France during World War I."

Final Answer: Marie Curie, France.

DeepSieve combines structured filtering and semantic reasoning, correctly narrowing down candidates using SQL and then using RAG to identify the scientific contribution and wartime location. Pure RAG fails to apply the 19th-century constraint and misattributes the discovery.

B Modular sources

Table 4 summarizes the modular source modules available in DeepSieve. Each source is represented as a (Tool, Corpus) pair with an associated profile string. These profiles serve as natural language descriptors visible to the LLM router during routing decisions. The table illustrates the heterogeneity across source types (structured, unstructured, external APIs, and semi-structured logs), as well as their access modes (e.g., SQL, RAG, JSON, API). This modular abstraction enables plug-and-play source integration and facilitates fine-grained control over per-subquestion retrieval.

C Prompt Examples

DeepSieve's effectiveness stems from carefully designed prompts that implement our core methodology. Each prompt type serves a distinct purpose

in the reasoning pipeline while maintaining consistency with our framework's principles.

Decomposition Prompt

Decomposition Prompt

You are a question planner. Decompose the complex question into a sequence of atomic questions that can be answered using a single fact each.
Original Query: `Who succeeded the Prime Minister that resigned during the Brexit vote?`
Decomposed Subquestions: 1. Who was the UK Prime Minister during the Brexit vote? 2. Who succeeded that Prime Minister?
Only output the list of subquestions in order. Do not include explanations.

Design Rationale: This prompt operationalizes our information sieving principle (Section 2.1) by forcing atomic decomposition. The constrained format ensures each subquestion targets exactly one retrievable fact while preserving dependency relationships. The "no explanations" requirement minimizes token usage for downstream processing.

Routing Prompt

Routing Prompt

You are a routing assistant. Your task is to decide whether a query should be answered using which tool-data pair.
Available Pairs: - `local`: people and entity-specific information. - `global`: general world knowledge including geography, history, etc.
Query: `What state is Montebello located in?`
Please output only one word: `local` or `global`. Do not explain your choice.

Design Rationale: Implements our source-aware routing through extreme output simplification. The binary choice format minimizes token overhead while enforcing discrete source selection. This aligns perfectly with our tool-corpus abstraction and enables efficient routing decisions.

Reflexion Prompt

Reflexion Prompt

You are a reflective reasoning agent. The previous attempt failed to find a valid answer. Try to rephrase or redirect the subquestion.
Failed Query: `What is one of the stars of "The Newcomers" known for?` Failed Result: `Dano is an indie film actor.` (not grounded)
Try to reflect and generate a new query that might work better.
Reflected Subquestion: `What is Chris Evans known for?`

Design Rationale: Embodies our iterative refinement approach. The prompt structure guides the LLM to: (1) recognize retrieval failures, (2) preserve original intent, and (3) eliminate ambiguous references - all while maintaining the atomic question constraint.

Fusion Prompt

Fusion Prompt

You are an answer synthesis agent. Given the original question and a list of subquestion-answer pairs, generate a final answer. Be concise and faithful to the evidence.
Original Question: `What country is the birthplace of Erik Hort a part of?`
Sub-QA Chain: 1. Who was born in Montebello? → Erik Hort 2. What state is Montebello in? → New York 3. What country is New York in? → United States
Final Answer: `United States`

Design Rationale: Implements our evidence aggregation protocol (Section 2.4). The prompt enforces three key requirements: (1) traceability to subanswers, (2) conflict resolution, and (3) minimal elaboration - ensuring final outputs remain grounded in the retrieved evidence.

D Source Profiles for Routing

To support per-subquery source routing, each corpus in DeepSieve is associated with a router-visible profile string that summarizes its scope and intended use. These profiles are automatically injected into the routing prompt (see Section **??**) to

guide source selection. Below we list the local/-global profile definitions used for each benchmark.

MuSiQue

The MuSiQue benchmark utilizes two complementary knowledge sources with distinct characteristics:

Local Profile (Entity-Specific)

This knowledge base focuses on specific named entities such as People (e.g., artists, politicians), Organizations (e.g., companies, institutions), Locations (e.g., cities, countries), Events (e.g., historical or sports events), and Works (e.g., books, films, artworks). It is ideal for entity-centric or biographical questions.

Global Profile (Contextual Knowledge)

This knowledge base contains more general background or contextual information that is not specific to any single named entity. It is more comprehensive and serves as a general fallback when the entity-specific source cannot answer the query. Select this profile when the query is not related to any specific entity in local profile.

2WikiMultiHopQA

For 2WikiMultiHopQA, we maintain the same local/global dichotomy but with adaptations for its unique multi-hop reasoning requirements:

Local Profile (Entity-Centric)

This corpus includes documents related to People, Organizations, Locations, Events, and notable Works. It emphasizes named entities and is well-suited for questions involving specific persons, places, or creations.

Global Profile (General Knowledge)

This corpus provides general-purpose knowledge and covers a wide range of background facts. It should be used when queries are not focused on specific named entities or require general reasoning. Select this profile when the query is not related to any specific entity in local profile.

HotpotQA

HotpotQA's profiles are specially designed to handle its bridge and comparison questions, with clear separation between focused and diffuse knowledge:

Local Profile (Focused Facts)

This local knowledge base contains concise factual descriptions about specific named entities. It includes individual biographies, locations, media works, historical figures, organizations, and structured encyclopedic entries. Each entry focuses on a single topic and offers concentrated factual coverage.
Examples of content:

- Biographies of notable people
- Descriptions of countries, cities, or institutions
- Synopses of TV shows, films, or novels
- Explanations of historical events or wars

Global Profile (Diffuse Knowledge)

This global knowledge base consists of diverse, loosely categorized facts and contextual information. It often includes edge cases, composite references, or entries that lack clear structural focus. These documents may span multiple entities, vague topics, or ambiguous scopes, making them harder to ground precisely.
Examples of content:

- Mentions of multiple entities without clear subject
- Abstract summaries or uncommon references
- Indirect relationships between people and places

E Pseudocode Implementation

The core logic is presented in two parts: the main control pipeline and the key LLM-driven helper functions. The main pipeline is designed as a highly modular and configurable controller. Its behavior is governed by a configuration object C, which allows for the systematic enabling or dis-

abling of key components such as decomposition, routing, and reflexion.

Algorithm: The DeepSieve Pipeline

Input: Query Q, Set of source modules S, Config C
Output: Final answer $\hat{A}$

```
M ← InitializeMemory()        // memory for results & failures
if C.decompose then
    {q_i} ← Decompose(Q)      // Stage I: Decompose
else
    {q_i} ← {Q}               // RAG-only setting
for each subquery q_i in execution order do
    q_i^actual ← SubstituteVariables(q_i, M)
    a_i ← null;  is_success ← false;  j ← 0

    while j < C.max_reflexion_attempts and not is_success do
        if C.use_routing then
            s_i ← Route(q_i^actual, S, M.failures)  // Stage II: Route
        else
            s_i ← S_merged // RAG-only: use merged
        a_candidate ← s_i.Execute(q_i^actual)  // retrieve or tool call
        a_i, is_success ← ExtractAnswer(a_candidate)  // Stage III: Observe
        if not is_success and C.use_reflexion then
            M.LogFailure(q_i^actual, s_i)  // reflect on failure
        j ← j + 1
    M.LogSuccess(q_i, a_i)
Â ← Fuse(Q, M)                // Stage IV: Fuse all answers
return Â
```

This design is a significant advantage as it was crucial for conducting the ablation studies presented in our experiments. The algorithm's versatility is highlighted by the if C.use_routing block, which shows how the framework can execute its primary logic for heterogeneous sources or gracefully degrade to a standard "RAG-only" baseline operating on a single merged source.

Algorithm: Helper Functions for DeepSieve

```
Function Decompose(Q):
    prompt ← CreateDecompositionPrompt(Q)
    response ← LLM(prompt)
    return ParseSubqueryDAG(response)

Function Route(q, S, Fails):
    prompt ← CreateRoutingPrompt(q, S, Fails)
    response ← LLM(prompt)
    return SelectSourceFrom(response, S)

Function ExtractAnswer(a_candidate):
    prompt ← CreateExtractionPrompt(a_candidate)
    response ← LLM(prompt)
    (a, s) ← ParseAnswerAndSuccess(response)
    return (a, s)

Function Fuse(Q, M):
    prompt ← CreateFusionPrompt(Q, M.successes)
    response ← LLM(prompt)
    return ParseFinalAnswer(response)
```

The reasoning process begins with an explicit planning step, where the Decompose function generates a structured Directed Acyclic Graph (DAG) of subqueries. A key strength of this approach is that it makes the reasoning plan transparent, machine-readable, and capable of modeling the complex, non-linear dependencies required to solve multi-hop questions. The execution of this plan is both dynamic and adaptive. For each subquery, the Route function intelligently selects a knowledge source, while the while loop embodies the Reflexion stage, providing a robust self-correction mechanism. If a retrieval is unsuccessful, the failure is logged to memory, allowing the system to attempt a different route on the next iteration.

This entire workflow is powered by the helper functions in Algorithm **??**, which abstract the core cognitive tasks into distinct, prompt-driven LLM calls. By encapsulating operations like Decompose and Fuse into modular functions, the framework adheres to the "LLM-as-a-Controller" paradigm. This has the advantage of making the system's logic transparent and easy to modify, as the behavior of each stage can be fine-tuned simply by refining

its corresponding prompt without altering the core control flow code.

F Baseline Details

We provide additional details for all baselines compared in our experiments.

ColBERTv2 (Santhanam et al., 2022). A late-interaction dense retriever with efficient token-level matching. We use the open-sourced checkpoint trained on MS MARCO. It serves as the base retriever for IRCoT and HippoRAG.

IRCoT (Trivedi et al., 2023). A strong multi-hop QA system that interleaves retrieval and CoT-style reasoning. We use their official implementation with ColBERTv2 as the retriever, following the original two-stage design.

HippoRAG (Gutierrez et al., 2024). A neuro-inspired long-term memory RAG system that builds a hierarchical memory graph over retrieved content. We follow their open implementation and retrieval setup with ColBERTv2.

RAPTOR (Sarthi et al., 2024). A recent RAG framework using recursive abstraction and document-level graph indexing. We follow their standard OpenIE + KG construction pipeline and include their released knowledge graphs for fair comparison.

ReAct (Yao et al., 2023). An agent-style system that integrates reasoning and action via thought-action-observation loops. Our version uses the retrieval-augmented ReAct setup as implemented in their codebase.

ReWOO (Xu et al., 2023). An improved CoT method that trains workers and orchestrators to coordinate on reasoning and observation generation. We follow their implementation with the same planner-answerer split.

Reflexion (Madaan et al., 2023). A framework that iteratively refines answers based on self-evaluation. We implement a retrieval-augmented version where the model reflects after each retrieval failure.

Chain-of-Thought (CoT) (Wei et al., 2022). A classic prompting baseline using few-shot reasoning examples without external retrieval. We use standard CoT prompts and apply them to each dataset individually.

G Dataset Descriptions

We describe the three multi-hop QA datasets used in our experiments:

MuSiQue (Trivedi et al., 2022). A challenging benchmark designed to test multi-hop and compositional reasoning. Each question requires aggregating facts across multiple Wikipedia passages. To reduce spurious correlations, MuSiQue provides both contrastive distractors and minimal context chains. We use the full MuSiQue-Answerable (MuSiQue-Full) version.

2WikiMultiHopQA (Fu et al., 2021). A clean and diverse multi-hop QA dataset built from Wikipedia entity pairs, where each question involves reasoning over two connected entities. It supports entity linking and is less noisy than HotpotQA. We follow prior work in using the dev split (1,000 samples) for evaluation.

HotpotQA (Yang et al., 2018). A widely-used benchmark featuring bridge and comparison questions that require multi-hop reasoning over Wikipedia. Though popular, it is known to contain noise in both questions and support passages. We use the distractor setting, which includes 10 retrieved passages per question with only one or two relevant ones.

All datasets are preprocessed using IRCoT's corpus construction pipeline, with supporting and distractor passages combined to form a flat retrieval index. The same set of questions is used across all methods for fair comparison.

H Experiment Setting Details

To ensure consistency and reproducibility across all stages of reasoning, we adopt a unified LLM inference setup for all core components in DeepSieve—including decomposition, routing, reflexion, and final answer fusion. Each of these modules invokes a chat-based language model via HTTP API following the OpenAI-compatible `/chat/completions` protocol.

We use **deterministic decoding** by setting the generation temperature to 0.0, ensuring that the same input yields the same output across runs. This is critical for stability during iterative retrieval and reflection chains, and avoids stochastic drift in multi-hop inference.

By default, we use either `gpt-4o` or `deepseek-chat` as the backend model. These

Table 4: Modular source modules in DeepSieve and their router-visible profiles. Each source advertises a profile string that summarizes its domain and retrieval capability. The router relies on these profiles to select the most suitable source per subquestion.

Source Name	Type	Access Mode	Profile for Router Prompt
PersonnelDB	Structured	SQL	Personnel records such as employee names, roles, and office locations. Use this source if the question involves internal or private company information.
Wikipedia	Unstructured	RAG corpus	This is a general-purpose encyclopedia covering broad world knowledge, including named entities, places, and public facts.
Google API	External Tool	API	This is a live web search API for retrieving real-time or geo-specific public information, such as current location, weather, or map-based data.
LogJSON	Semi-structured	JSON	This source holds semi-structured JSON logs from user chat or activity history. Use it when the query refers to personal history or past interactions.

are selected based on experiment configuration or availability constraints. All prompts are constructed as single-turn queries, sent as a single `user`-role message without tool calling or system-level instructions. The response is expected in one pass, typically in JSON or structured text format.

HTTP client settings. All LLM requests are issued through a pooled session with adapter-level retries. The client is configured as follows:

- **Temperature:** 0.0 (greedy decoding, no randomness)
- **Max retries:** 3
- **Timeout per request:** 60 seconds
- **Retry strategy:** Exponential backoff with delay 2^i for the i-th retry
- **Error conditions triggering retry:** HTTP status 429 (rate limit), 408 (timeout), 500–504 (server errors)

This robust client configuration minimizes disruptions during long-running experiments, where thousands of LLM queries may be issued in sequence. It also handles transient failures gracefully, such as momentary rate limits or backend latency spikes.

Implementation details. The entire LLM calling logic is encapsulated in a lightweight utility script:

> `utils/llm_call.py` implements `call_openai_chat(prompt, api_key, model, base_url)`, which handles inference, retry logic, and timeout handling in a self-contained function.

This modular setup ensures that all DeepSieve modules, including planning, tool routing, and failure recovery, use a consistent and reliable interface to external language models. It also facilitates easy swapping between providers (e.g., OpenAI, DeepSeek, local vLLM) by simply changing the base URL and model string without modifying any internal pipeline logic.

I Mudular Experiment: Can Routing Preserve Expert Performance?

Motivation. A key motivation for our work is the need to handle queries spanning heterogeneous knowledge sources—such as structured databases and unstructured corpora—that cannot be merged into a unified index. In real-world systems, queries may originate from diverse domains and require reasoning over non-coalescent sources like SQL tables and general-purpose document collections. To address this, DeepSieve introduces a routing module that decides, per query, which tool-data pair to invoke. But does this routing step degrade the performance of expert tools?

Setup. We design a diagnostic evaluation to test whether DeepSieve's routing module preserves the upper-bound performance of expert tools when operating over distinct question types. We construct a test set consisting of two disjoint groups:

- **Circle group**: factual questions about famous people (e.g., birth dates, professions), best answered via SQL queries.
- **Reg group**: general knowledge questions involving entities, events, and context, suitable for RAG-style text retrieval.

Each group is paired with a dedicated expert:

- **Circle → SQLTool**: executes structured queries over a database of historical figures.
- **Reg → SimpleRAG**: answers queries using a standard vector retriever over text corpus.

Settings. We compare two evaluation configurations:

- **Setting 1 (Expert Oracle Routing)**: Each question is routed to its designated expert tool—Circle queries go to `SQLTool`, Reg queries go to `SimpleRAG`. This reflects the performance of a perfect oracle router.
- **Setting 2 (DeepSieve Routing)**: All queries are passed through DeepSieve's LLM-based router, which must decide which tool to use without access to question type labels.

Table 5: Deepsieve vs. Expert Oracle performance (Exact Match | F1). The underline means that the results are calculated in an ideal setting which is not feasible in real scene.

Setting	SQL	RAG	Overall
Expert Oracle (S1)	45.0 | 46.2	32.4 | 41.8	<u>38.7</u> | <u>44.0</u>
DeepSieve (S2)	50.8 | 52.0	48.3 | 59.1	35.6 | 40.1

Results.

Analysis. Setting 1 provides an upper bound under ideal conditions, where each question is perfectly matched with its corresponding expert tool. In contrast, Setting 2 reflects a realistic deployment scenario, where the system must decide the tool on the fly, without access to ground-truth labels or perfect routing hints.

Despite the performance drop in Setting 2 compared with the ideal setting, DeepSieve achieves reasonable performance in the absence of any oracle guidance. This validates the viability of routing-based reasoning under heterogeneous data regimes.

Why Routing is Necessary. While Setting 1 performs better, it assumes prior knowledge of the best tool per question—an unrealistic expectation in practice. More importantly, real-world questions often span multiple sources, making joint reasoning across SQL and RAG data essential. Since these sources cannot be merged, a routing mechanism is indispensable.

This experiment demonstrates:

- Routing is critical for modular, multi-source reasoning.
- DeepSieve can support this with acceptable trade-offs in performance.
- Even under idealized upper bounds, the performance gap is not catastrophic.