Table 11: Comparison RAPTOR and RAPTOR-K.

Method	MultihopQA		Quality	PopQA	
	Accuracy	Recall	Accuracy	Accuracy	Recall
RAPTOR	56.064	**44.832**	**56.997**	62.545	27.304
RAPTOR-K	**56.768**	44.208	54.567	**64.761**	**28.469**

A ADDITIONAL EXPERIMENTS

A.1 Results on specific QA tasks

In this Section, we present the additional results on the specific QA tasks.

▶ **Exp.1. Comparison RAPTOR-K and RAPTOR.** We compare RAPTOR-K with RAPTOR on the first three datasets, and present results in Table 11. We observe that RAPTOR-K achieves comparable or even better performance than RAPTOR.

▶ **Exp.2. Token costs of graph and index building.** We report the token costs of building four types of graphs across HotpotQA and ALCE datasets in Figure 14. Recall that the token cost for an LLM call consists of two parts: the prompt token, which accounts for the tokens used in providing the input, and the completion part, which includes the tokens generated by the model as a response. Here, we report the token costs for prompt and completion on HotpotQA and ALCE in Figure 14(c) to (d), and show the results on other datasets in Figure 15. We conclude that, regardless of the graph type, the prompt part always incurs higher token costs than the completion part.

▶ **Exp.3. Evaluation of the generation costs.** In this experiment, we evaluate the time and token costs for each method in specific QA tasks. Specifically, we report the average time and token costs for each query across all datasets in Table 12, the conclusions are consistent with those reported in our manuscript. As shown in Figure 16, we present the average token costs for prompt tokens and completion tokens across all questions in all specific QA datasets. We can observe that the running time of each method is highly proportional to the completion token costs, which aligns with the computational paradigm of the Transformer architecture.

▶ **Exp.4. Detailed analysis for RAPTOR and LGraphRAG.** Our first analysis about RAPTOR aims to explain why RAPTOR outperforms VanillaRAG. Recall that in RAPTOR, for each question Q, it retrieves the top-k items across the entire tree, meaning the retrieved items may originate from different layers. That is, we report the proportion of retrieved items across different tree layers in Table 13. As we shall see, for the MultihopQA and MusiqueQA datasets, the proportion of retrieved high-level information (i.e., items not from leaf nodes) is significantly higher than in other datasets. For datasets requiring multi-hop reasoning to answer questions, high-level information plays an essential role. This may explain why RAPTOR outperforms VanillaRAG on these two datasets.

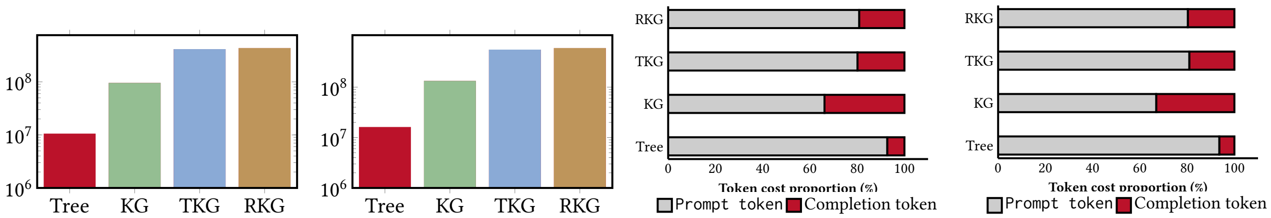
We then conduct a detailed analysis of LGraphRAG on complex questions in specific QA datasets by modifying its retrieval methods or element types. By doing this, we create three variants of LGraphRAG, and we present the detailed descriptions for each variant in Table 14. Here, VGraphRAG-CC introduces a new retrieval strategy. Unlike LGraphRAG, it uses vector search to retrieve the top-k elements (i.e., chunks or communities) from the vector database. Eventually, we evaluate their performance on the three complex QA datasets and present the results in Table 15.

▶ **Exp.5. Effect of the chunk size.** Recall that our study includes some datasets that are pre-split by the export annotator. To investigate this impact, we re-split the corpus into multiple chunks based on token size for these datasets instead of using their original chunks. Here, we create three new datasets from HotpotQA, PopQA, and ALCE, named HotpotAll, PopAll, and ALCEAll, respectively.

For each dataset, we use Original to denote its original version and New chunk to denote the version after re-splitting. We report the results of graph-based RAG methods on both the original and new version datasets in Figure 17, we can see that: (1) The performance of all methods declines, mainly because rule-based chunk splitting (i.e., by token size) fails to provide concise information as effectively as expert-annotated chunks. (2) Graph-based methods, especially those relying on TKG and RKG, are more sensitive to chunk quality. This is because the graphs they construct encapsulate richer information, and coarse-grained chunk splitting introduces potential noise within each chunk. Such noise can lead to inaccurate extraction of entities or relationships and their corresponding descriptions, significantly degrading the performance of these methods. (3) As for token costs, all methods that retrieve chunks incur a significant increase due to the larger chunk size in New chunk compared to Original, while other methods remain stable. These findings highlight that chunk segmentation quality is crucial for the overall performance of all RAG methods.

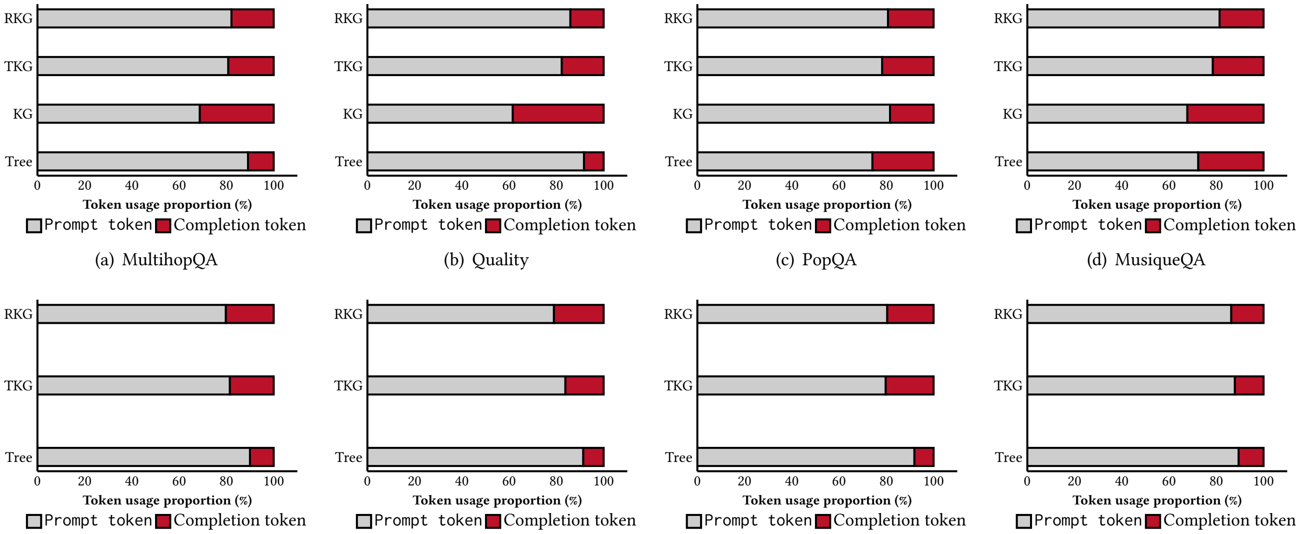
We evaluate all 12 RAG methods under chunk sizes of 600, 1200, and 2400 tokens, as shown in Table 16, and further select the top-performing eight methods to analyze how their performance varies with chunk size, as shown in Figure 18. We observe that the results across different settings remain largely consistent with the conclusions drawn using the default chunk size of 1200 in our submitted manuscript. Specifically, we make the following observations and analysis: (1) For simple QA datasets (e.g., PopAll and HotpotAll), smaller chunk sizes generally yield better performance. This is because such questions often require information that is directly available in a single chunk or two. Smaller chunks provide more focused and precise context, improving answer accuracy. (2) Across all chunk sizes, HippoRAG and RAPTOR perform best on simple QA tasks, while RAPTOR and LGraphRAG outperform others on complex QA datasets (e.g., MultihopQA, ALCEAll). These observations align with the results reported in our manuscript. (3) The effect of chunk size varies by task type: performance on simple QA tasks is highly sensitive to chunk size, while for complex QA tasks, performance remains relatively stable across different chunk sizes. This is because complex questions typically require reasoning across multiple chunks, making them less dependent on individual chunk granularity.

▶ **Exp.6. Effect of the LLM backbones.** For each system, we evaluate multiple LLM backbones, including Llama-3-8B, Qwen-2.5-32B, Llama-3-70B, and GPT-4o-mini. These cover three open-source models ranging from small to large scales, as well as one proprietary yet strong GPT-family model. In this experiment, we fix the chunk size at 1200 tokens. As shown in Table 17, we evaluate each RAG method and the ZeroShot baseline on the MultihopQA and ALCEAll datasets using different LLM backbones. Based on the results, we can see that: (1) Stronger models generally yield better performance, especially in the ZeroShot setting, which most directly reflects the inherent capabilities of the underlying LLM. (2) We observe that the three variants of LightRAG, LLightRAG, GLightRAG, and HLightRAG as well as LGraphRAG, achieve significant performance improvements when using more powerful LLMs.



(a) HotpotQA (b) ALCE (c) The token costs on HotpotQA (d) The token costs on ALCE

Figure 14: Token cost of graph building on specific QA datasets.



(a) MultihopQA (b) Quality (c) PopQA (d) MusiqueQA

(e) Mix (f) Agriculture (g) CS (h) Legal

Figure 15: Proportion of the token costs for prompt and completion in graph building stage across all datasets.

Table 12: The average time and token costs of all methods on specific QA datasets.

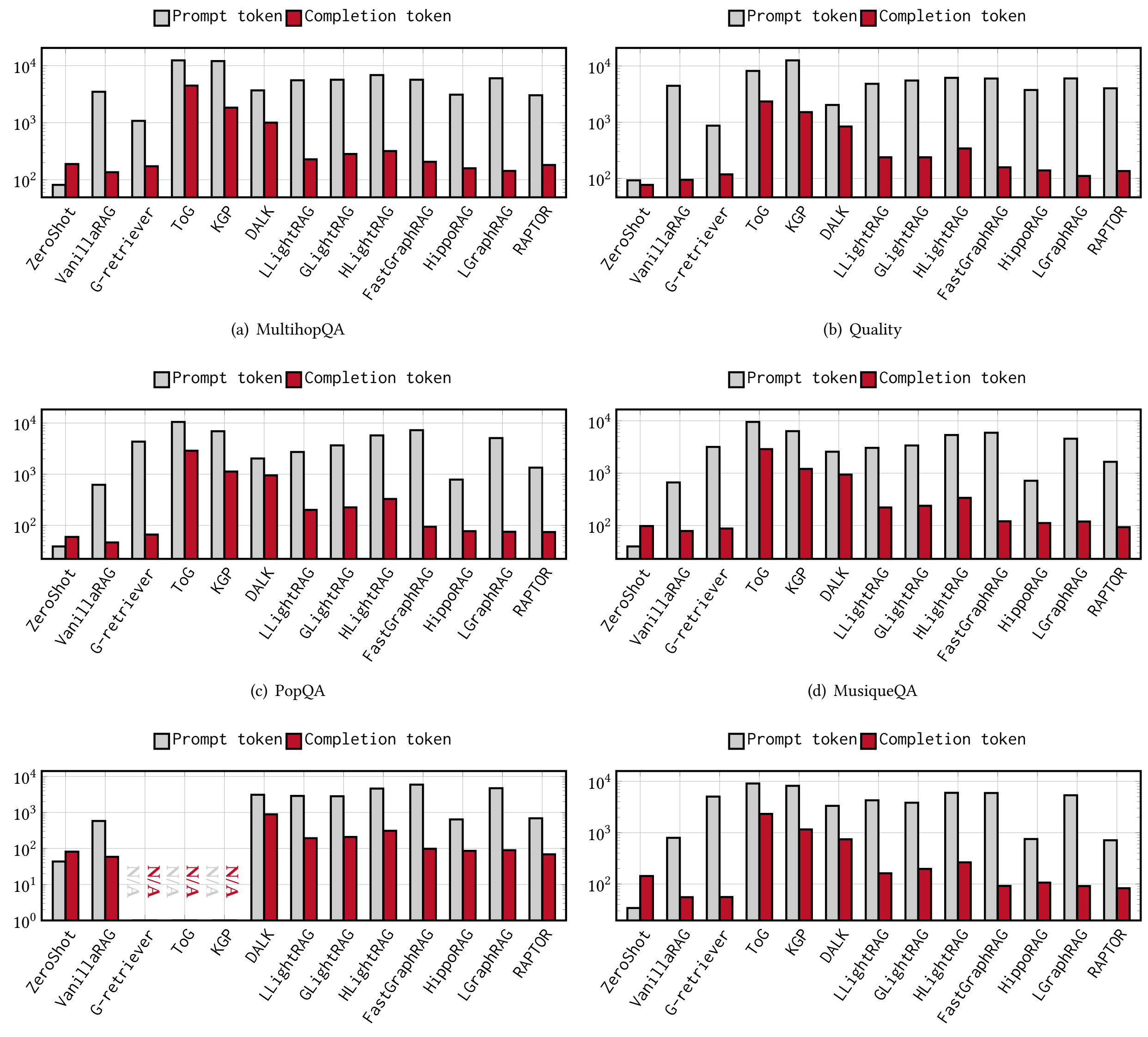
Method	MultihopQA		Quality		PopQA		MusiqueQA		HotpotQA		ALCE	
	time	token	time	token	time	token	time	token	time	token	time	token
ZeroShot	3.23 s	270.3	1.47 s	169.1	1.17 s	82.2	1.73 s	137.8	1.51 s	125.0	2.41 s	177.2
VanillaRAG	2.35 s	3,623.4	2.12 s	4,502.0	1.41 s	644.1	1.31 s	745.4	1.10 s	652.0	1.04 s	849.1
G-retriever	6.87 s	1,250.0	5.18 s	985.5	37.51 s	3,684.5	31.21 s	3,260.5	—	—	101.16 s	5,096.1
ToG	69.74 s	16,859.6	37.03 s	10,496.4	42.02 s	11,224.2	53.55 s	12,480.8	—	—	34.94 s	11,383.2
KGP	38.86 s	13,872.2	35.76 s	14,092.7	37.49 s	6,738.9	39.82 s	7,555.8	—	—	105.09 s	9,326.6
DALK	28.03 s	4,691.5	13.23 s	2,863.6	16.33 s	2,496.5	17.48 s	3,510.9	21.33 s	3,989.7	17.04 s	4,071.9
LLightRAG	19.28 s	5,774.1	15.76 s	5,054.5	10.71 s	2,447.5	13.95 s	3,267.6	13.94 s	3,074.2	10.34 s	4,427.9
GLightRAG	18.37 s	5,951.5	15.97 s	5,747.3	12.10 s	3,255.6	15.20 s	3,260.8	13.95 s	3,028.7	13.02 s	4,028.1
HLightRAG	19.31 s	7,163.2	21.49 s	6,492.5	17.71 s	5,075.8	20.93 s	5,695.3	19.58 s	4,921.7	16.55 s	6,232.3
FastGraphRAG	7.17 s	5,874.8	3.48 s	6,138.9	13.25 s	6,157.0	15.19 s	6,043.5	28.71 s	6,029.8	25.82 s	6,010.9
HippoRAG	3.46 s	3,261.1	3.03 s	3,877.6	2.32 s	721.3	2.69 s	828.4	3.12 s	726.4	2.94 s	858.2
LGraphRAG	2.98 s	6,154.9	3.77 s	6,113.7	1.72 s	4,325.2	2.66 s	4,675.7	2.05 s	4,806.2	2.11 s	5,441.1
RAPTOR	3.18 s	3,210.0	2.46 s	4,140.7	1.36 s	1,188.3	1.85 s	1,742.9	1.48 s	757.6	1.54 s	793.6

Table 14: Descriptions of the different variants of LGraphRAG.

Name	Retrieval elements	New retrieval strategy
LGraphRAG	Entity, Relationship, Community, Chunk	✗
GraphRAG-ER	Entity, Relationship	✗
GraphRAG-CC	Community, Chunk	✗
VGraphRAG-CC	Community, Chunk	✓
VGraphRAG	Entity, Relationship, Community, Chunk	✓

Table 13: Proportion of retrieved nodes across tree layers.

Layer	MultihopQA	Quality	PopQA	MusiqueQA	HotpotQA	ALCE
0	59.3%	76.8%	76.1%	69.3%	89.7%	90.6%
1	27.5%	18.7%	16.5%	28.1%	9.5%	8.8%
> 1	13.2%	4.5%	7.4%	2.6%	0.8%	0.6%



(a) MultihopQA

(b) Quality

(c) PopQA

(d) MusiqueQA

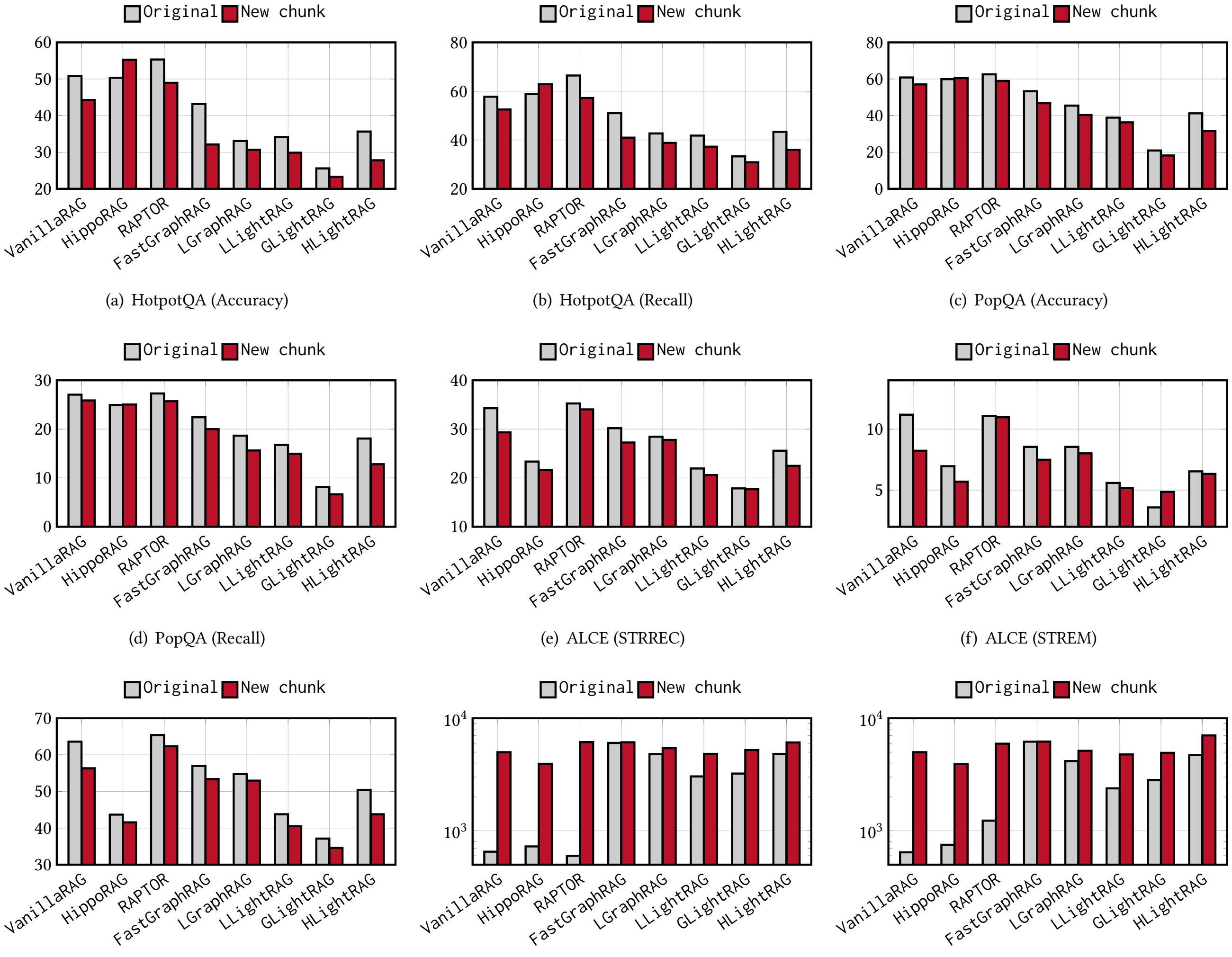
(e) HotpotQA

(f) ALCE

Figure 16: Token costs for prompt and completion tokens in the generation stage across all datasets.

Table 15: Comparison of our newly designed methods on specific datasets with complex questions.

Dataset	Metric	ZeroShot	VanillaRAG	LGraphRAG	RAPTOR	GraphRAG-ER	GraphRAG-CC	VGraphRAG-CC	VGraphRAG
MultihopQA	Accuracy	49.022	50.626	55.360	**56.064**	52.739	52.113	55.203	**59.664**
	Recall	34.526	36.918	**50.429**	44.832	45.113	43.770	46.750	**50.893**
MusiqueQA	Accuracy	1.833	17.233	12.467	**24.133**	11.200	13.767	22.400	**26.933**
	Recall	5.072	27.874	23.996	**35.595**	22.374	25.707	35.444	**40.026**
ALCE	STRREC	15.454	34.283	28.448	35.255	26.774	35.366	**37.820**	**41.023**
	STREM	3.692	11.181	8.544	11.076	7.5949	11.920	**13.608**	**15.401**
	STRHIT	30.696	63.608	54.747	65.401	52.743	64.662	**68.460**	**71.835**



(a) HotpotQA (Accuracy)

(b) HotpotQA (Recall)

(c) PopQA (Accuracy)

(d) PopQA (Recall)

(e) ALCE (STRREC)

(f) ALCE (STREM)

(g) ALCE (STRHIT)

(h) HotpotQA (Token)

(i) PopQA (Token)

Figure 17: Effect of chunk quality on the performance of specific QA tasks.

Table 16: **Comparison of methods on different datasets under different chunk sizes, where Purple denotes the best result, and Orange denotes the best result excluding the best one.**

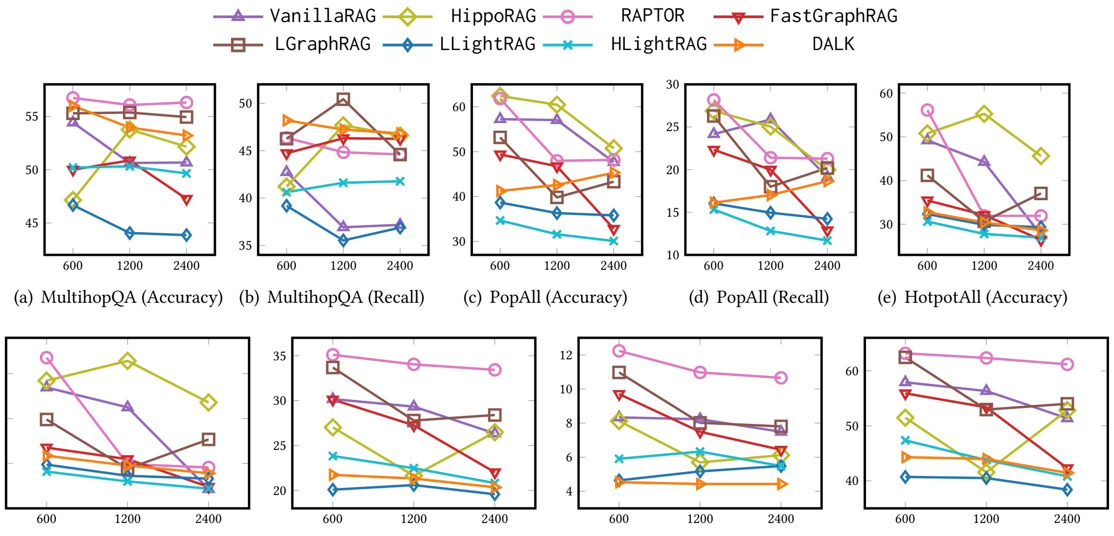
Method	Chunk Size	MultihopQA		PopAll		HotpotAll		ALCEAll		
		Accuracy	Recall	Accuracy	Recall	Accuracy	Recall	STRREC	STREM	STRHIT
VanillaRAG	600	54.421	42.740	57.255	24.171	49.190	56.935	30.174	8.333	57.911
	1200	50.626	36.918	57.041	25.877	44.254	52.511	29.334	8.228	56.329
	2400	50.665	37.172	47.677	19.122	27.553	34.293	26.350	7.490	51.371
G-retriever	600	44.953	47.322	19.014	6.895	18.882	27.041	10.506	2.532	21.414
	1200	42.019	43.116	20.157	6.567	19.098	27.262	13.061	2.848	26.160
	2400	41.667	36.611	20.229	6.763	27.958	36.940	14.822	2.954	30.591
ToG	600	42.214	37.823	49.035	22.642	–	–	16.240	3.376	33.439
	1200	41.941	38.435	49.750	22.767	–	–	15.909	3.059	33.755
	2400	41.667	36.611	44.961	20.374	–	–	14.822	2.954	30.591
KGP	600	46.831	34.714	41.101	18.064	–	–	26.799	7.806	51.793
	1200	48.161	36.272	41.887	18.498	–	–	27.551	7.806	52.637
	2400	48.083	36.486	40.815	18.060	–	–	26.370	6.540	51.477
DALK	600	55.986	**48.202**	41.243	16.131	32.766	41.737	21.734	4.536	44.304
	1200	53.952	47.232	42.602	17.024	30.416	39.544	21.327	4.430	43.987
	2400	53.208	46.829	45.318	18.651	28.633	37.826	20.350	4.430	41.456
LLightRAG	600	46.675	39.190	38.671	16.040	32.334	39.753	20.084	4.641	40.717
	1200	44.053	35.528	36.312	14.947	29.876	37.258	20.594	5.169	40.506
	2400	43.858	36.878	35.811	14.230	29.336	36.621	19.576	5.485	38.397
GLightRAG	600	47.222	36.885	20.801	6.860	24.419	31.852	18.319	5.063	36.814
	1200	48.474	38.365	18.227	6.643	23.285	30.888	17.686	4.852	34.599
	2400	48.592	41.213	14.653	5.890	19.665	27.566	13.493	3.376	26.688
HLightRAG	600	50.196	40.626	34.668	15.326	30.632	38.194	23.833	5.907	47.363
	1200	50.313	41.613	31.594	12.812	27.796	36.010	22.475	6.329	43.776
	2400	49.648	41.775	30.093	11.683	26.958	34.387	20.816	5.485	40.823
FastGraphRAG	600	50.000	44.736	49.392	22.326	35.467	43.534	30.116	9.705	55.907
	1200	52.895	44.278	46.748	19.996	32.118	40.966	27.258	7.490	53.376
	2400	47.261	46.251	32.809	12.879	26.445	34.800	22.020	6.435	42.300
HippoRAG	600	47.144	41.210	**62.401**	**26.892**	50.783	58.454	27.025	8.122	51.477
	1200	53.760	47.671	60.472	25.041	**55.267**	**62.862**	21.633	5.696	41.561
	2400	52.152	46.601	50.751	19.986	45.624	53.597	26.477	6.118	52.848
LGraphRAG	600	55.282	46.267	53.181	26.292	41.194	49.801	33.692	**10.971**	**62.447**
	1200	55.360	**50.429**	39.814	17.998	30.686	38.824	27.785	8.017	52.954
	2400	54.930	44.588	43.317	20.185	37.061	45.366	28.398	7.806	54.008
RAPTOR	600	**56.729**	46.358	**61.830**	**28.176**	**56.132**	**63.584**	**35.111**	**12.236**	**63.186**
	1200	56.064	44.832	47.963	21.399	31.983	39.864	**34.044**	**10.971**	62.342
	2400	**56.299**	44.610	48.177	21.289	31.983	39.122	33.432	10.654	61.181

This can be attributed to their reliance on Rich Knowledge Graphs and Textual Knowledge Graphs, where stronger LLMs contribute to the construction of higher-quality graphs. (3) HippoRAG shows notably superior performance when using GPT-4o-mini compared to other LLM backbones. We attribute this to GPT-4o-mini's ability to extract more accurate entities from the question and to construct higher-quality knowledge graphs, thereby improving the retrieval of relevant chunks and the final answer accuracy. (4) Regardless of the LLM backbone, our proposed method VGraphRAG consistently achieves the best performance, demonstrating the advantages of our proposed unified framework.

▶ **Exp.7. The size of graph.** For each dataset, we report the size of five types of graphs in Table 18. We observe that PG is typically denser than other types of graphs, as they connect nodes based on

Table 17: The specific QA performance comparison of graph-based RAG methods with different LLM backbones.

Method	LLM backbone	MultihopQA		ALCEAll		
		Accuracy	Recall	STRREC	STREM	STRHIT
ZeroShot	Llama-3-8B	49.022	34.256	15.454	3.692	30.696
	Qwen-2.5-32B	45.070	33.332	30.512	10.127	56.118
	Llama-3-70B	55.908	52.987	31.234	7.170	61.920
	GPT-4o-mini	59.546	48.322	34.965	10.232	66.245
VanillaRAG	Llama-3-8B	50.626	36.918	29.334	8.228	56.329
	Qwen-2.5-32B	56.299	47.660	39.490	14.873	69.937
	Llama-3-70B	56.768	49.127	34.961	9.810	68.038
	GPT-4o-mini	59.311	47.941	35.735	10.127	68.249
G-retriever	Llama-3-8B	42.019	43.116	13.061	2.848	26.160
	Qwen-2.5-32B	43.075	33.864	34.678	13.608	60.338
	Llama-3-70B	50.430	50.144	24.575	5.907	49.051
	GPT-4o-mini	56.534	46.427	31.681	7.068	62.764
ToG	Llama-3-8B	41.941	38.435	15.909	3.059	33.755
	Qwen-2.5-32B	34.390	35.566	19.167	4.430	39.662
	Llama-3-70B	42.762	38.495	18.449	3.270	38.819
	GPT-4o-mini	41.862	30.247	28.405	7.068	56.962
KGP	Llama-3-8B	48.161	36.272	27.551	7.806	52.637
	Qwen-2.5-32B	61.463	57.340	40.608	14.873	71.097
	Llama-3-70B	55.008	47.878	35.420	10.549	68.379
	GPT-4o-mini	63.146	55.789	38.015	12.764	69.620
DALK	Llama-3-8B	53.952	47.232	21.327	4.430	43.987
	Qwen-2.5-32B	32.003	20.158	16.653	4.430	33.333
	Llama-3-70B	60.524	55.086	28.980	5.696	60.338
	GPT-4o-mini	66.980	57.687	31.813	8.333	62.236
LLightRAG	Llama-3-8B	44.053	35.528	20.594	5.169	40.506
	Qwen-2.5-32B	48.552	45.387	31.549	9.916	59.177
	Llama-3-70B	57.081	54.510	28.636	8.228	56.013
	GPT-4o-mini	52.113	38.923	38.446	15.084	33.456
GLightRAG	Llama-3-8B	48.474	38.365	17.686	4.852	34.599
	Qwen-2.5-32B	52.582	48.236	27.961	9.599	53.165
	Llama-3-70B	55.986	51.713	23.553	5.807	45.992
	GPT-4o-mini	55.125	47.899	39.095	**15.506**	67.933
HLightRAG	Llama-3-8B	50.313	41.613	22.475	6.329	43.776
	Qwen-2.5-32B	53.678	51.403	34.168	10.971	63.819
	Llama-3-70B	57.081	54.510	29.548	8.228	57.911
	GPT-4o-mini	55.829	46.424	**41.334**	**15.506**	71.730
FastGraphRAG	Llama-3-8B	52.895	44.278	27.258	7.490	53.376
	Qwen-2.5-32B	46.088	50.370	31.387	10.021	59.388
	Llama-3-70B	54.069	55.787	35.658	12.236	65.612
	GPT-4o-mini	66.080	57.007	23.521	8.228	42.827
HippoRAG	Llama-3-8B	53.760	47.671	21.633	5.696	41.561
	Qwen-2.5-32B	48.083	40.488	37.419	13.397	66.245
	Llama-3-70B	57.277	57.736	32.904	9.916	32.534
	GPT-4o-mini	**67.723**	55.482	39.274	12.447	**72.046**
LGraphRAG	Llama-3-8B	55.360	50.429	27.785	8.017	52.954
	Qwen-2.5-32B	49.531	52.113	35.406	12.553	63.924
	Llama-3-70B	58.060	55.390	34.256	10.232	66.561
	GPT-4o-mini	65.415	50.216	36.890	11.287	69.304
RAPTOR	Llama-3-8B	56.064	44.832	34.044	10.971	62.342
	Qwen-2.5-32B	60.485	56.359	39.267	13.924	70.359
	Llama-3-70B	63.028	**61.042**	37.286	12.236	68.671
	GPT-4o-mini	60.603	51.521	29.770	8.017	58.861
VGraphRAG	Llama-3-8B	59.664	50.893	35.213	11.603	64.030
	Qwen-2.5-32B	57.277	55.151	39.234	14.557	69.831
	Llama-3-70B	67.567	**68.445**	37.576	12.447	69.198
	GPT-4o-mini	**68.193**	56.564	**43.963**	**18.038**	**74.473**

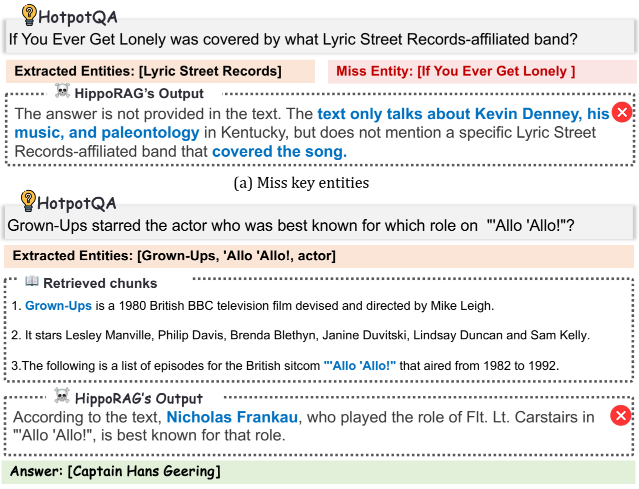


(a) MultihopQA (Accuracy) (b) MultihopQA (Recall) (c) PopAll (Accuracy) (d) PopAll (Recall) (e) HotpotAll (Accuracy)

(f) HotpotAll (Recall) (g) ALCEAll (STRREC) (h) ALCEAll (STREM) (i) ALCEAll (STRHIT)

Figure 18: Effect of chunk size.

shared entity relationships, where each node represents a chunk in PG. In fact, the probability of two chunks sharing at least a few entities is quite high, leading to a high graph density (i.e., the ratio of edges to nodes), sometimes approaching a clique (fully connected graph). In contrast, KG, TKG, and RKG are much sparser since they rely entirely on LLMs to extract nodes and edges. This sparsity is primarily due to the relatively short and incomplete outputs typically generated by LLMs, which miss considerable potential node-edge pairs. Interestingly, the size or density of the constructed graph has not shown a strong correlation with the final performance of graph-based RAG methods. This observation motivates us to explore a method for evaluating the quality of the constructed graph before using it for LLM-based question answering.


HotpotQA

If You Ever Get Lonely was covered by what Lyric Street Records-affiliated band?

Extracted Entities: [Lyric Street Records] Miss Entity: [If You Ever Get Lonely]

HippoRAG's Output

The answer is not provided in the text. The text only talks about Kevin Denney, his music, and paleontology in Kentucky, but does not mention a specific Lyric Street Records-affiliated band that covered the song.

(a) Miss key entities

HotpotQA

Grown-Ups starred the actor who was best known for which role on "'Allo 'Allo!"?

Extracted Entities: [Grown-Ups, 'Allo 'Allo!, actor]

Retrieved chunks

1. Grown-Ups is a 1980 British BBC television film devised and directed by Mike Leigh.

2. It stars Lesley Manville, Philip Davis, Brenda Blethyn, Janine Duvitski, Lindsay Duncan and Sam Kelly.

3.The following is a list of episodes for the British sitcom "'Allo 'Allo!" that aired from 1982 to 1992.

HippoRAG's Output

According to the text, Nicholas Frankau, who played the role of Flt. Lt. Carstairs in "'Allo 'Allo!", is best known for that role.

Answer: [Captain Hans Geering]

(b) Irrelevant chunks

Figure 19: The failure cases of HippoRAG.

▶ **Exp.8. Failure cases of graph-based RAG methods.** We would like to clarify that the results presented in our paper represent the average performance across all questions within each dataset. Thus, not all graph-based RAG methods consistently outperform the baseline VanillaRAG on every question. We conduct a detailed failure case analysis focusing on the top-performing methods. Specifically, we examine why RAPTOR, HippoRAG, RAPTOR and LGraphRAG sometimes fall short in specific QA tasks. Please refer to the detailed analysis provided below.

▶ **The key failure reasons of HippoRAG:**

(1) *Incorrect or incomplete entity extraction from the question.* Consider the example in Figure 19(a), where HippoRAG extracts the entity "Lyric Street Records" from the question and then applies Personalized PageRank based on this entity to retrieve relevant chunks. However, to answer this question correctly, two key entities are required: "Lyric Street Records" and "If You Ever Get Lonely". Since HippoRAG fails to extract the latter, it retrieves chunks that are insufficient to provide a correct answer.

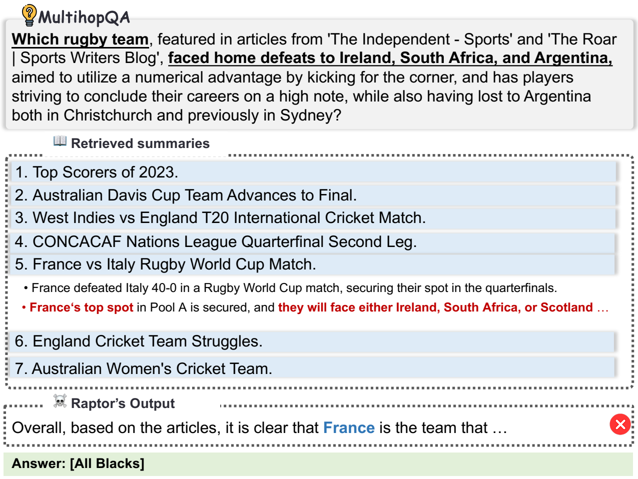
(2) *Retrieval of irrelevant chunks by Personalized PageRank.* In another example shown in Figure 19(b), HippoRAG correctly extracts the relevant entities ["Grown-ups", "Allo 'Allo!", "actor"] from the question. However, its chunk retriever strategy—Personalized PageRank—tends to favor chunks where these entities appear frequently, regardless of whether the content is semantically relevant to the question. As a result, the retrieved chunks may not align with the actual intent of the question, leading to an incorrect final answer.

▶**The key failure reason of Raptor:**

- *Low-quality cluster summaries.* As illustrated in Figure 20, Raptor retrieves cluster summaries generated from groups of similar chunks. However, chunks within the same cluster may mention various loosely related facts that are topically similar but not logically unified. For example, the summary "Top Scorers of 2023" in Figure 20 contains some loosely related facts about scorers in 2023, which are too general to provide a precise answer. When summarizing such content, the LLM tends to produce generic or fragmented summaries

Table 18: The size of each graph type across all datasets.

Dataset	Tree		PG		KG		TKG		RKG	
	# of vertices	# of edges	# of vertices	# of edges	# of vertices	# of edges	# of vertices	# of edges	# of vertices	# of edges
MultihopQA	2,053	2,052	1,658	564,446	35,953	37,173	12,737	10,063	18,227	12,441
Quality	1,862	1,861	1,518	717,468	28,882	30,580	10,884	8,992	13,836	9,044
PopQA	38,325	38,324	32,157	3,085,232	260,202	336,676	179,680	205,199	188,946	215,623
MusiqueQA	33,216	33,215	29,898	3,704,201	228,914	295,629	153,392	183,703	149,125	188,149
HotpotQA	73,891	73,890	66,559	13,886,807	511,705	725,521	291,873	401,693	324,284	436,362
ALCE	99,303	99,302	89,376	22,109,289	610,925	918,499	306,821	475,018	353,989	526,486
Mix	719	718	1,778	1,225,815	28,793	34,693	7,464	2,819	7,701	3,336
MultihopSum	2,053	2,052	N/A	N/A	N/A	N/A	12,737	10,063	18,227	12,441
Agriculture	2,156	2,155	N/A	N/A	N/A	N/A	15,772	7,333	17,793	12,600
CS	2,244	2,243	N/A	N/A	N/A	N/A	10,175	6,560	12,340	8,692
Legal	5,380	5,379	N/A	N/A	N/A	N/A	15,034	10,920	16,565	17,633



MultihopQA

Which rugby team, featured in articles from 'The Independent - Sports' and 'The Roar | Sports Writers Blog', **faced home defeats to Ireland, South Africa, and Argentina,** aimed to utilize a numerical advantage by kicking for the corner, and has players striving to conclude their careers on a high note, while also having lost to Argentina both in Christchurch and previously in Sydney?

Retrieved summaries

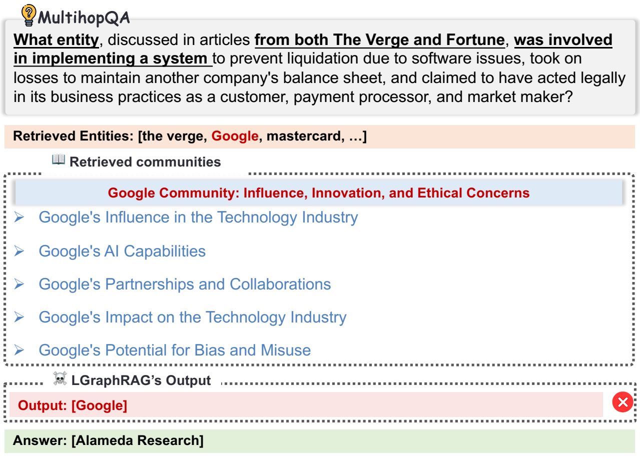
1. Top Scorers of 2023.
2. Australian Davis Cup Team Advances to Final.
3. West Indies vs England T20 International Cricket Match.
4. CONCACAF Nations League Quarterfinal Second Leg.
5. France vs Italy Rugby World Cup Match.
   - France defeated Italy 40-0 in a Rugby World Cup match, securing their spot in the quarterfinals.
   - **France's top spot** in Pool A is secured, and **they will face either Ireland, South Africa, or Scotland** …
6. England Cricket Team Struggles.
7. Australian Women's Cricket Team.

Raptor's Output

Overall, based on the articles, it is clear that France is the team that …

Answer: [All Blacks]

Figure 20: The failure case of Raptor.



MultihopQA

What entity, discussed in articles **from both The Verge and Fortune**, **was involved in implementing a system** to prevent liquidation due to software issues, took on losses to maintain another company's balance sheet, and claimed to have acted legally in its business practices as a customer, payment processor, and market maker?

Retrieved Entities: [the verge, Google, mastercard, …]

Retrieved communities

Google Community: Influence, Innovation, and Ethical Concerns

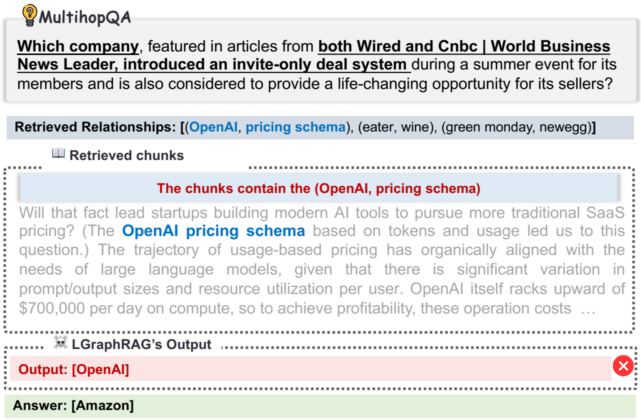
- Google's Influence in the Technology Industry
- Google's AI Capabilities
- Google's Partnerships and Collaborations
- Google's Impact on the Technology Industry
- Google's Potential for Bias and Misuse

LGraphRAG's Output

Output: [Google]

Answer: [Alameda Research]

Figure 21: The failure case of LGraphRAG (I).



MultihopQA

Which company, featured in articles from **both Wired and Cnbc | World Business News Leader, introduced an invite-only deal system** during a summer event for its members and is also considered to provide a life-changing opportunity for its sellers?

Retrieved Relationships: [(OpenAI, pricing schema), (eater, wine), (green monday, newegg)]

Retrieved chunks

The chunks contain the (OpenAI, pricing schema)

Will that fact lead startups building modern AI tools to pursue more traditional SaaS pricing? (The **OpenAI pricing schema** based on tokens and usage led us to this question.) The trajectory of usage-based pricing has organically aligned with the needs of large language models, given that there is significant variation in prompt/output sizes and resource utilization per user. OpenAI itself racks upward of $700,000 per day on compute, so to achieve profitability, these operation costs …

LGraphRAG's Output

Output: [OpenAI]

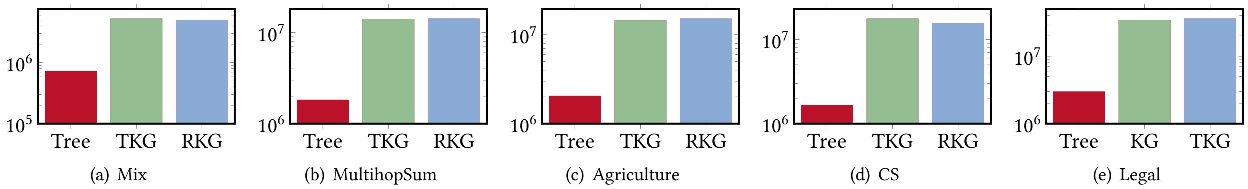
Answer: [Amazon]

Figure 22: The failure case of LGraphRAG (II).

that fail to capture the key information required to answer the question.

▶The key failure reasons of LGraphRAG:

- *Irrelevant community reports retrieved by* Entity *operator.* Consider the example in Figure 21, where LGraphRAG first extracts entities such as ["The Verge", "Google", "Mastercard", · · ·] from the question. It then applies the Entity operator to retrieve communities whose reports contain these entities. Among them, communities with frequent mentions of "Google" are prioritized. However, these retrieved communities turn out to be irrelevant to the actual question, as the method relies solely on surface-level entity frequency while ignoring semantic relevance.
- *Irrelevant chunks retrieved by* Occurrence *operator.* Consider the example in Figure 22, where LGraphRAG extracts relationships such as [("OpenAI", "pricing schema"), ("Eater", "wine"), ('greenmonday", "newegg")] from the question. It then applies the Occurrence operator to retrieve chunks that contain the relationship ("OpenAI", "pricing schema") with high frequency. Based on these chunks, LGraphRAG generates the incorrect answer "OpenAI". The key reason is that the retrieved chunks, despite their frequent mentions of certain relationships, are not semantically relevant to the question. The method relies on co-occurrence frequency rather than actual contextual relevance.



(a) Mix (b) MultihopSum (c) Agriculture (d) CS (e) Legal

Figure 23: Token cost of the graph building on abstract QA datasets.

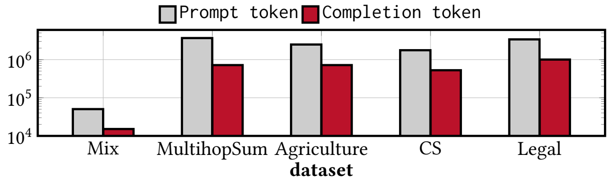


Figure 24: Token cost of index construction in abstract QA.

A.2 Results on abstract QA tasks

In this subsection, we present the addition results on abstract QA tasks.

▶ **Exp.1. Token costs of graph and index building.** The token costs of the graph and index building across all abstract QA datasets are shown in Figures 23 and 24 respectively. The conclusions are highly similar to the Exp.2 in Section 7.2.

▶ **Exp.2. Effect of chunk size.** We report the performance of various RAG methods on abstract QA tasks under different chunk sizes in Figures 25 to 27. Our key observations are as follows: (1) The performance of GGraphRAG remains stable across different chunk sizes, likely due to its use of the Map-Reduce strategy for final answer synthesis, which mitigates the influence of chunk granularity. (2) In contrast, methods like FastGraphRAG and VanillaRAG show greater variance across chunk sizes, as their performance relies heavily on the granularity of individual chunks—smaller chunks tend to provide more precise information, directly impacting retrieval and generation quality. (3) Regardless of chunk size, RAPTOR and GGraphRAG consistently achieve the best performance, reaffirming our earlier conclusion that high-level structural information is essential for abstract QA tasks.

Additionally, we evaluate our newly proposed method CheapRAG across different chunk sizes. As shown in Figure 28, CheapRAG generally outperforms the five baselines. Notably, under the 600-token setting, CheapRAG surpasses GGraphRAG in more cases. We attribute this to the higher precision of smaller chunks, which enhances the effectiveness of semantic similarity-based retrieval in CheapRAG, compared to the entity frequency-based retrieval strategy used in GGraphRAG.

▶ **Exp.3. Effect of LLM backbone.** We evaluate all methods that support abstract QA on the MultihopSum dataset, using different LLM backbones. Results for Llama-3-8B are shown in Figure 26, while Figures 29 to 31 present results for the other models. We observe that, across different backbones, the performance of each method on abstract QA tasks remains relatively stable—especially when compared to the fluctuations seen in specific QA tasks. We note that all methods still lag behind GGraphRAG, further highlighting that community-level information is particularly beneficial for abstract QA tasks. Moreover, we compare our newly proposed method, CheapRAG, with five strong baselines under different LLM backbones. As shown in Figure 32, CheapRAG exhibits remarkable performance improvements as the model capacity increases. Notably, under the GPT-4o-mini backbone, CheapRAG achieves near-universal wins across all evaluated cases, clearly demonstrating its strong generalization ability and effectiveness.

▶ **Exp.4. More analysis.** We have further analyzed the permetric results across different methods, and summarize the key insights as follows:

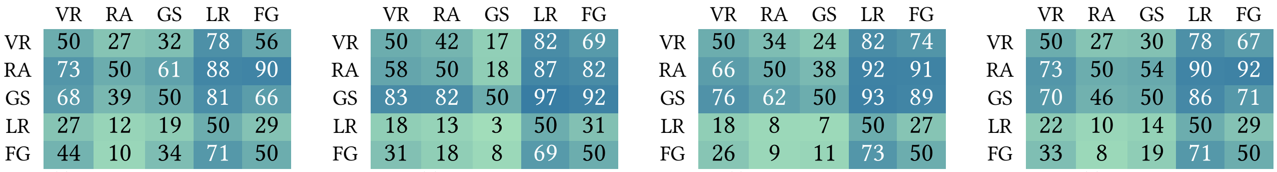
- *Comprehensiveness:* GGraphRAG consistently achieves the highest scores, highlighting the strength of community-level retrieval in capturing global context. By grouping semantically related content, communities help reduce fragmented evidence and support more holistic answers.
- *Diversity:* Both RAPTOR and GGraphRAG perform strongly by aggregating information across multiple clusters or communities. This enables the generation of responses that span diverse subtopics while maintaining relevance.
- *Empowerment:* GGraphRAG and LightRAG jointly lead on this metric. Their retrieval strategies incorporate structured elements—entities, relations, and keywords—that provide concrete grounding for the model to generate actionable, role-relevant responses. This better supports practical decision-making in activity-centered QA.
- *Overall:* GGraphRAG consistently ranks first, with RAPTOR typically second. This highlights the advantage of leveraging high-level summarized information—such as community reports and cluster-level chunks—with the former generally proving more effective. Additionally, the results support the effectiveness of the Map-Reduce mechanism in filtering out irrelevant information during retrieval.

A.3 Evaluation metrics

This section outlines the metrics used for evaluation.

• **Metrics for specific QA Tasks.** We use accuracy as the evaluation metric, based on whether the gold answers appear in the model's generated outputs, rather than requiring an exact match, following the approach in [4, 57, 73]. This choice is motivated by the uncontrollable nature of LLM outputs, which often makes it difficult to achieve exact matches with standard answers. Similarly, we prefer recall over precision as it better reflects the accuracy of the generated responses.

• **Metrics for abstract QA Tasks.** Building on existing work, we use an LLM to generate abstract questions, as shown in Figure 34. We adopt 125 questions following the prior works, such as GraphRAG [15], LightRAG [24], and FastGraphRAG [19]. In their setup, the number 125 comes from generating N=5 user roles, each with N associated tasks, and for each (user, task) pair, N abstract questions—yielding $5 \times 5 \times 5 = 125$ questions per dataset. We follow this standard for consistency and comparability across studies. The reasons of selecting GPT-4o are twofold: (1) at the time



(a) Comprehensiveness

	VR	RA	GS	LR	FG
VR	50	27	32	78	56
RA	73	50	61	88	90
GS	68	39	50	81	66
LR	27	12	19	50	29
FG	44	10	34	71	50

(b) Diversity

	VR	RA	GS	LR	FG
VR	50	42	17	82	69
RA	58	50	18	87	82
GS	83	82	50	97	92
LR	18	13	3	50	31
FG	31	18	8	69	50

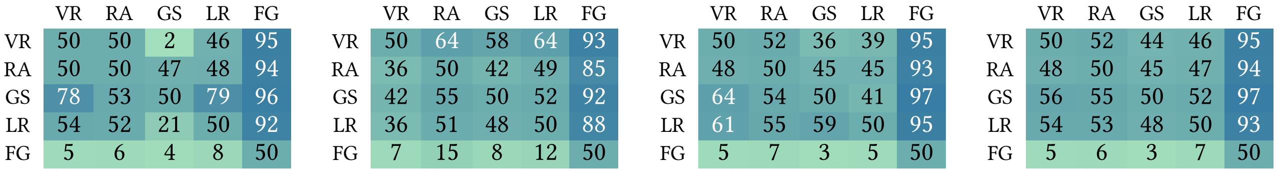
(c) Empowerment

	VR	RA	GS	LR	FG
VR	50	34	24	82	74
RA	66	50	38	92	91
GS	76	62	50	93	89
LR	18	8	7	50	27
FG	26	9	11	73	50

(d) Overall

	VR	RA	GS	LR	FG
VR	50	27	30	78	67
RA	73	50	54	90	92
GS	70	46	50	86	71
LR	22	10	14	50	29
FG	33	8	19	71	50

Figure 25: The abstract QA results on the MultihopSum dataset (chunk size = 600).



(a) Comprehensiveness

	VR	RA	GS	LR	FG
VR	50	50	2	46	95
RA	50	50	47	48	94
GS	78	53	50	79	96
LR	54	52	21	50	92
FG	5	6	4	8	50

(b) Diversity

	VR	RA	GS	LR	FG
VR	50	64	58	64	93
RA	36	50	42	49	85
GS	42	55	50	52	92
LR	36	51	48	50	88
FG	7	15	8	12	50

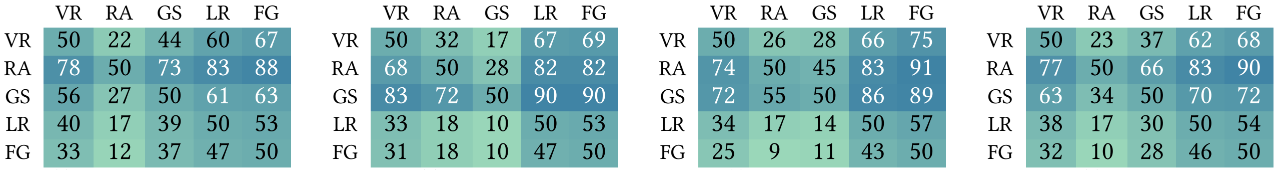
(c) Empowerment

	VR	RA	GS	LR	FG
VR	50	52	36	39	95
RA	48	50	45	45	93
GS	64	54	50	41	97
LR	61	55	59	50	95
FG	5	7	3	5	50

(d) Overall

	VR	RA	GS	LR	FG
VR	50	52	44	46	95
RA	48	50	45	47	94
GS	56	55	50	52	97
LR	54	53	48	50	93
FG	5	6	3	7	50

Figure 26: The abstract QA results on the MultihopSum dataset (chunk size = 1200).



(a) Comprehensiveness

	VR	RA	GS	LR	FG
VR	50	22	44	60	67
RA	78	50	73	83	88
GS	56	27	50	61	63
LR	40	17	39	50	53
FG	33	12	37	47	50

(b) Diversity

	VR	RA	GS	LR	FG
VR	50	32	17	67	69
RA	68	50	28	82	82
GS	83	72	50	90	90
LR	33	18	10	50	53
FG	31	18	10	47	50

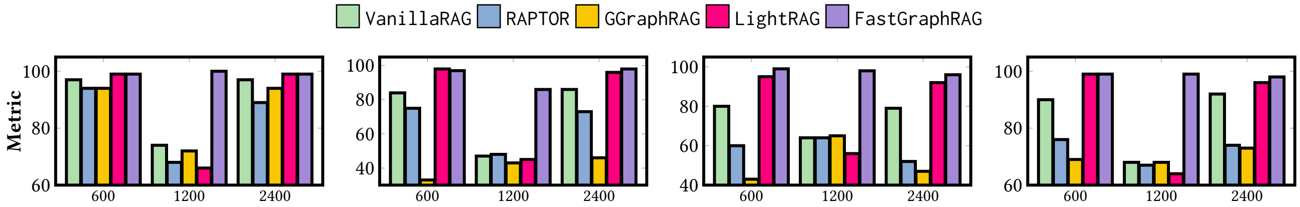
(c) Empowerment

	VR	RA	GS	LR	FG
VR	50	26	28	66	75
RA	74	50	45	83	91
GS	72	55	50	86	89
LR	34	17	14	50	57
FG	25	9	11	43	50

(d) Overall

	VR	RA	GS	LR	FG
VR	50	23	37	62	68
RA	77	50	66	83	90
GS	63	34	50	70	72
LR	38	17	30	50	54
FG	32	10	28	46	50

Figure 27: The abstract QA results on the MultihopSum dataset (chunk size = 2400).



(a) Comprehensiveness (b) Diversity (c) Empowerment (d) Overall

Figure 28: Comparison of CheapRAG and other methods under different chunk sizes on the MultihopSum dataset.

when conducted our experiments, GPT-4o was one of the most advanced LLM, demonstrating strong zero-shot capabilities and superior performance in long-context understanding compared to other models, and (2) it provides the highest fluency, coherence, and factual consistency, ensuring that the generated questions are both challenging and realistic. This effectively supports our exploration of real-world data sensemaking scenarios.

Defining ground truth for abstract questions, especially those involving complex high-level semantics, presents significant challenges. To address this, we adopt an LLM-based multi-dimensional comparison method, inspired by [15, 24], which evaluates comprehensiveness, diversity, empowerment, and overall quality. We use a robust LLM, specifically GPT-4o, to rank each baseline in comparison to our method. The evaluation prompt used is shown in Figure 35. These four metrics are defined [15, 19, 24] as follows:

- *Comprehensiveness.* How much detail does the answer provide to cover all aspects and details of the question?
- *Diversity.* How varied and rich is the answer in providing different perspectives and insights on the question?
- *Empowerment.* How well does the answer help the reader understand and make informed judgments about the topic?
- *Overall.* Select an overall winner based on these categories.

To better illustrate these dimensions, Figure 36 presents examples of both good and bad answers with respect to comprehensiveness, diversity, and empowerment.

Head-to-head comparison. To evaluate abstract QA tasks by head-to-head comparison using an LLM evaluator, selecting four target metrics capturing qualities that are desirable for abstract questions. The answer to an abstract question is not a collection of details from specific texts, but rather a high-level understanding of the dataset's contents relevant to the query. A good response to an abstract question should perform well across the following four metrics, including "Comprehensiveness", "Diversity", "Empowerment", and "Overall".

Head-to-head win results illustrate the relative performance among different methods. Each value indicates the percentage of test cases where the row method outperforms the column method - higher values indicate better performance. For example, Figure 37 demonstrates an example head-to-head result under the "comprehensive" dimension on the Mix dataset. The value 30 in the first row and third column indicates that VanillaRAG outperforms GGraphRAG in 30% of cases, suggesting that VanillaRAG performs

	VR	RA	GS	LR	FG
VR	50	17	9	14	62
RA	83	50	24	55	86
GS	91	76	50	77	93
LR	86	45	23	50	91
FG	38	14	7	9	50

(a) Comprehensiveness

	VR	RA	GS	LR	FG
VR	50	22	9	22	73
RA	78	50	26	60	86
GS	91	74	50	83	95
LR	78	40	17	50	90
FG	27	14	5	10	50

(b) Diversity

	VR	RA	GS	LR	FG
VR	50	26	12	26	74
RA	74	50	24	63	90
GS	88	76	50	84	94
LR	74	37	16	50	91
FG	26	10	6	9	50

(c) Empowerment

	VR	RA	GS	LR	FG
VR	50	19	10	18	68
RA	81	50	24	61	88
GS	90	76	50	80	93
LR	82	39	20	50	91
FG	32	12	7	9	50

(d) Overall

Figure 29: The abstract QA results on the MultihopSum dataset (LLM backbone = Qwen-2.5-32B).

	VR	RA	GS	LR	FG
VR	50	10	18	16	70
RA	90	50	49	68	96
GS	82	51	50	63	90
LR	84	32	37	50	94
FG	30	4	10	6	50

(a) Comprehensiveness

	VR	RA	GS	LR	FG
VR	50	15	1	19	67
RA	85	50	5	68	91
GS	99	95	50	99	99
LR	81	32	1	50	88
FG	33	9	1	12	50

(b) Diversity

	VR	RA	GS	LR	FG
VR	50	13	4	29	81
RA	87	50	20	74	97
GS	96	80	50	91	99
LR	71	26	9	50	93
FG	19	3	1	7	50

(c) Empowerment

	VR	RA	GS	LR	FG
VR	50	12	6	18	78
RA	88	50	30	76	97
GS	94	70	50	83	99
LR	82	24	17	50	95
FG	22	3	1	5	50

(d) Overall

Figure 30: The abstract QA results on the MultihopSum dataset (LLM backbone = Llama-3-70B).

	VR	RA	GS	LR	FG
VR	50	66	4	12	44
RA	34	50	5	9	31
GS	96	95	50	81	94
LR	88	91	19	50	82
FG	56	69	6	18	50

(a) Comprehensiveness

	VR	RA	GS	LR	FG
VR	50	61	6	15	54
RA	39	50	6	14	39
GS	94	94	50	85	95
LR	85	86	15	50	87
FG	46	61	5	13	50

(b) Diversity

	VR	RA	GS	LR	FG
VR	50	79	8	31	74
RA	21	50	4	14	40
GS	92	96	50	92	95
LR	69	86	8	50	86
FG	26	60	5	14	50

(c) Empowerment

	VR	RA	GS	LR	FG
VR	50	72	4	13	60
RA	28	50	4	10	37
GS	96	96	50	90	95
LR	87	90	10	50	87
FG	40	63	5	13	50

(d) Overall

Figure 31: The abstract QA results on the MultihopSum dataset (LLM backbone = GPT-4o-mini).

	Com.	Div.	Emp.	Overall
VR	74	47	64	68
RA	68	48	64	67
GS	72	43	65	68
LR	66	45	56	64
FG	100	86	98	99

(a) LLM backbone = Llama-3-8B

	Com.	Div.	Emp.	Overall
VR	99	99	99	99
RA	87	96	91	93
GS	90	81	60	82
LR	99	99	97	99
FG	99	99	99	99

(b) LLM backbone = Qwen-2.5-32B

	Com.	Div.	Emp.	Overall
VR	98	97	91	96
RA	93	77	61	73
GS	95	40	76	88
LR	98	94	76	88
FG	99	99	98	99

(c) LLM backbone = Llama-3-70B

	Com.	Div.	Emp.	Overall
VR	99	99	99	99
RA	99	99	99	99
GS	64	90	72	80
LR	93	99	96	98
FG	99	99	99	99

(d) LLM backbone = GPT-4o-mini

Figure 32: Comparison of our newly designed method on MultihopSum dataset.

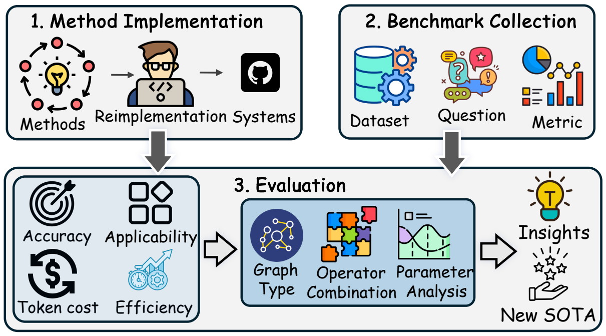


Figure 33: Workflow of our empirical study.

worse than GGraphRAG on this metric. Based on the overall comparison in the figure, the relative ranking of the five methods is: 1. HLightRAG (LR), 2. GGraphRAG (GS), 3. VanillaRAG (VR), 4. RAPTOR (RA), 5. FastGraphRAG (FG).

A.4 Implementation details

In this subsection, we present more details about our system implementation. Specifically, we use HNSW [56] from Llama-index [55] (a well-known open-source project) as the default vector database for efficient vector search. In addition, for each method, we optimize efficiency by batching or parallelizing operations such as encoding nodes or chunks, and computing personalized page rank, among others, during the retrieval stage.

Why Llama-3-8B? Beyond the widespread use of Llama-3-8B in existing works, our choice is further motivated by the following considerations:

(1) *Strong Capability*: Llama-3-8B exhibits solid language understanding and reasoning abilities [14], which are crucial for RAG workflows. Its effectiveness in integrating retrieved content with internal knowledge enables accurate and contextually relevant response generation.

(2) *Practical Efficiency*: The 8B variant offers a favorable trade-off between performance and resource efficiency. It supports FP16 inference within 20 GB of GPU memory, allowing deployment on widely available hardware (e.g., a single 24 GB GPU).

(3) *Open source LLM*: As an open-source LLM, it can be deployed locally, saving the costs of paying for API calls required by proprietary models such as GPT-4o.

Workflow of our evaluation. We present the first open-source testbed for graph-based RAG methods, which (1) collects and reimplements 12 representative methods within a unified framework (as depicted in Section 3). (2) supports a fine-grained comparison over the building blocks of the retrieval stage with up to 100+ variants, and (3) provides a comprehensive evaluation over 11 datasets with various metrics in different scenarios, we summarize the workflow of our empirical study in Figure 33.

A.5 Criteria of selection dataset

Our work aims to systematically analyze various graph-based RAG methods and provide in-depth insights. To this end, we select datasets based on the following criteria:

1. *Widely used in the existing works:* All selected datasets are extensively used in the RAG and LLM research communities. For example, the HotpotQA dataset has been cited more than 2,800 times.
2. *Diverse corpus domains:* Our selected datasets cover a broad range of corpus domains. For instance, MultihopRAG consists of long English news articles; HotpotQA includes short passages from Wikipedia; and Quality comprises a diverse mixture of sources such as fiction from Project Gutenberg and articles from Slate magazine, with each document containing at least 2,000 tokens. For abstract questions, we include datasets spanning various domains such as agriculture, computer science, and legal texts.
3. *Diverse in task types:* Question answering (QA) tasks are typically categorized into abstract and specific questions [24], in which the specific questions can be further divided by complexity into simple and complex types [4, 72]. Specifically, simple questions typically require only one or two text chunks for an answer, without the need for multi-hop reasoning. In contrast, complex questions necessitate reasoning across multiple chunks, understanding implicit relations, and synthesizing information.

B MORE DISCUSSIONS.

B.1 New operators

Here, we introduce the operators that are not used in existing graph-based RAG methods but are employed in our newly designed state-of-the-art methods.

Chunk type. We include a new operator VDB of chunk type, which is used in our VGraphRAG method. This operator is the same as the chunk retrieval strategy of VanillaRAG.

Community type. We also include a new operator VDB of community type, retrieving the top-k communities by vector searching, where the embedding of each community is generated by encoding its community report.

B.2 More Lessons and Opportunities

In this subsection, we show the more lessons and opportunities learned from our study.

Lessons

▶ L6. For large datasets, both versions of the GraphRAG methods incur unacceptable token costs, as they contain a large number of communities, leading to high costs for generating community reports.

▶ L7. Regardless of whether the questions are specific or abstract, they all rely on an external corpus (i.e., documents). For such questions, merely using graph-structure information (nodes, edges, or subgraphs) is insufficient to achieve good performance.

▶ L8. Methods designed for knowledge reasoning tasks, such as DALK, ToG, and G-retriever, do not perform well on document-based QA tasks. This is because these methods are better suited for extracting reasoning rules or paths from well-constructed KGs. However, when KGs are built from raw text corpora, they may not accurately capture the correct reasoning rules, leading to suboptimal performance in document-based QA tasks.

▶ L9. The effectiveness of RAG methods is highly impacted by the relevance of the retrieved elements to the given question. That is, if the retrieved information is irrelevant or noisy, it may degrade the LLM's performance. When designing new graph-based RAG methods, it is crucial to evaluate whether the retrieval strategy effectively retrieves relevant information for the given question.

Opportunities

▶ O5. In real applications, external knowledge sources are not limited to text corpora; they may also include PDFs, HTML pages, tables, and other structured or semi-structured data. A promising future research direction is to explore graph-based RAG methods for heterogeneous knowledge sources.

▶ O6. An interesting future research direction is to explore more graph-based RAG applications. For example, applying graph-based RAG to scientific literature retrieval can help researchers efficiently extract relevant studies and discover hidden relationships between concepts. Another potential application is legal document analysis, where graph structures can capture case precedents and legal interpretations to assist in legal reasoning.

▶ O7. The users may request multiple questions simultaneously, but existing graph-based RAG methods process them sequentially. Hence, a promising future direction is to explore efficient scheduling strategies that optimize multi-query handling. This could involve batching similar questions or parallelizing retrieval.

▶ O8. Different types of questions require different levels of information, yet all existing graph-based RAG methods rely on fixed, predefined rules. How to design an adaptive mechanism that can address these varying needs remains an open question.

▶ O9. Existing methods do not fully leverage the graph structure; they typically rely on simple graph patterns (e.g., nodes, edges, or k-hop paths). Although GraphRAG adopts a hierarchical community structure (detecting by the Leiden algorithm), this approach does not consider node attributes, potentially compromising the quality of the communities. That is, determining which graph structures are superior remains an open question.

▶ O10. The well-known graph database systems, such as Neo4j [61] and Nebula [60], support transferring the corpus into a knowledge graph via LLM. However, enabling these popular systems to support the diverse operators required by various graph-based RAG methods presents an exciting opportunity.

B.3 Benefit of our framework

Our framework offers exceptional flexibility by enabling the combination of different methods at various stages. This modular design allows different algorithms to be seamlessly integrated, ensuring that each stage—such as *graph building*, and *retrieval&generation*—can be independently optimized and recombined. For example, methods like HippoRAG, which typically rely on KG, can easily be adapted to use RKG instead, based on specific domain needs.

In addition, our operator design allows for simple modifications—often just a few lines of code—to create entirely new graph-based RAG methods. By adjusting the retrieval stage or swapping components, researchers can quickly test and implement new strategies, significantly accelerating the development cycle of graph-based RAG methods.

The modular nature of our framework is further reinforced by the use of retrieval elements (such as node, relationship, or subgraph) coupled with retrieval operators. This combination enables us to easily design new operators tailored to specific tasks. For example, by modifying the strategy for retrieving given elements, we can create customized operators that suit different application scenarios.

By systematically evaluating the effectiveness of various retrieval components under our unified framework, we can identify the most efficient combinations of graph construction, indexing, and retrieval strategies. This approach enables us to optimize retrieval performance across a range of use cases, allowing for both the enhancement of existing methods and the creation of novel, state-of-the-art techniques.

Finally, our framework contributes to the broader research community by providing a standardized methodology to assess graph-based RAG approaches. The introduction of a unified evaluation testbed ensures reproducibility, promotes fair a benchmark, and facilitates future innovations in RAG-based LLM applications.

B.4 Limitations

In our empirical study, we put considerable effort into evaluating the performance of existing graph-based RAG methods from various angles. However, our study still has some limitations, primarily due to resource constraints. *(1) Token Length Limitation:* The primary experiments are conducted using Llama-3-8B with a token window size of 8k. This limitation on token length restricted the model's ability to process longer input sequences, which could potentially impact the overall performance of the methods, particularly in tasks that require extensive context. Larger models with larger token windows could better capture long-range dependencies and deliver more robust results. This constraint is a significant factor that may affect the generalizability of our findings. *(2) Limited Knowledge Datasets:* Our study did not include domain-specific knowledge datasets, which are crucial for certain applications. Incorporating such datasets could provide more nuanced insights and allow for a better evaluation of how these methods perform in specialized settings. *(3) Resource Constraints:* Due to resource limitations, the largest model we utilized is Llama-3-70B, and the entire paper consumes nearly 10 billion tokens. Running larger models, such as GPT-4o (175B parameters or beyond), would incur significantly higher costs, potentially reaching several hundred thousand dollars depending on usage. While we admit that introducing more powerful models could further enhance performance, the 70B model is already a strong choice, balancing performance and resource feasibility. That is to say, exploring the potential of even larger models in future work could offer valuable insights and further refine the findings. *(4) Prompt Sensitivity:* The performance of each method is highly affected by its prompt design. Due to resource limitations, we did not conduct prompt ablation studies and instead used the available prompts from the respective papers. Actually, a fairer comparison would mitigate this impact by using prompt tuning tools, such as DSPy [38], to customize the prompts and optimize the performance of each method.

These limitations highlight areas for future exploration, and overcoming these constraints would enable a more thorough and reliable evaluation of graph-based RAG methods, strengthening the findings and advancing the research.

B.5 Necessity of chunk splitting in RAG systems

Splitting the input corpus into smaller chunks is a necessary step for all RAG methods, including both non-graph and graph-based variants. The reasons can be summarized in three aspects below:

(1) *Input length limit of LLMs.* Every LLM has its own input length limitation. For example, ChatGPT-3.5 supports up to 4,096 tokens, while Llama-3-8B allows up to 8,192 tokens. A token is a basic unit of text (roughly a word or subword). However, real-world corpora often contain tens of thousands to millions of tokens—e.g., the ALCE dataset exceeds 13 million tokens—far beyond the processing capability of any existing LLM. Therefore, it is essential to split the corpus into smaller chunks.

(2) *Relevance filtering and noise reduction.* Even if the full corpus could fit into the LLM, chunking is still necessary because most queries only relate to a small part of the content. That is, given a query Q, inputting the entire corpus into the LLM introduces unnecessary noise. To alleviate this, RAG systems retrieve only the top-k relevant chunks, which helps filter out irrelevant content and improves both accuracy and efficiency.

(3) *Graph construction in graph-based RAG.* Chunking is equally essential and plays a critical role in all graph-based RAG methods. Below, we outline how different types of graphs rely on chunking:

- *Knowledge graph (KG), Textual Knowledge graph (TKG), and Rich Knowledge graph (RKG):* These graphs use LLMs to extract entities and relationships from the input corpus. Due to the token limitation of the LLM, the corpus must first be split into chunks. The LLM is then applied independently to each chunk, and the resulting outputs are aggregated to construct the complete graph. This follows a divide-and-conquer paradigm.
- *Passage graph (PG):* In the passage graph, each chunk is treated as a node, and an edge is added between two nodes if their corresponding chunks share at least a certain number of common entities. The graph is constructed by comparing entity overlap across all pairs of chunks.

In summary, chunking is not only necessary due to token limits but also integral to the design of RAG systems.

B.6 Comparison of graphs and indices

Indeed, it is difficult to define a single "best" graph, as each type of graph has its own advantages and disadvantages. To help clarify this, we summarize the key characteristics of each graph type in Table 19. For example, although tree can be constructed with minimal cost, it carries limited information. Methods such as RAPOTR, which are based on tree, demonstrate competitive performance under lightweight settings. However, they may fall short in complex QA and abstract QA tasks due to limited semantic coverage. In contrast, methods such as VGraphRAG and GGraphRAG, which are built upon TKG, benefit from richer entity and relationship representations. This enhanced expressiveness enables more accurate reasoning in information-intensive tasks. However, these advantages come at the cost of increased token consumption and longer construction time. Additionally, while PG avoids token usage during construction, it incurs significant time overhead due to exhaustive pairwise chunk comparisons.

Prompt for generating abstract questions

Prompt:
Given the following description of a dataset:
{description}
Please identify 5 potential users who would engage with this dataset. For each user, list 5 tasks they would perform with this dataset. Then, for each (user, task) combination, generate 5 questions that require a high-level understanding of the entire dataset.
Output the results in the following structure:

- **User 1: [user description]**
    - **Task 1: [task description]**
        - Question 1:
        - Question 2:
        - Question 3:
        - Question 4:
        - Question 5:
    - **Task 2: [task description]**
      ...
    - **Task 5: [task description]**
- **User 2: [user description]**
  ...
- **User 5: [user description]**
  ...

Note that there are 5 users and 5 tasks for each user, resulting in 25 tasks in total. Each task should have 5 questions, resulting in 125 questions in total. The Output should present the whole tasks and questions for each user.
Output:

Figure 34: The prompt for generating abstract questions.

Table 19: Comparison of different types of graph.

Graph	Entity Name	Entity Type	Entity Description	Relationship Name	Relationship Keyword	Relationship Description	Edge Weight	Token Consuming	Graph Size	Information Richness	Construction Time
Tree	✗	✗	✗	✗	✗	✗	✗	★	★	★★	★
PG	✗	✗	✗	✗	✗	✗	✗	N/A	★★★★	★	★★★★
KG	✓	✗	✗	✓	✗	✗	✓	★★	★★	★★★	★★
TKG	✓	✓	✓	✗	✗	✓	✓	★★★	★★★	★★★★	★★★
RKG	✓	✓	✓	✗	✓	✓	✓	★★★	★★★	★★★★★	★★★

Table 20: Comparison of different types of index.

Index	Index Size	Token Consuming	Construction Time
Node Index	★★	N/A	★
Relationship Index	★★★	N/A	★★
Community Index	★	★★★★	★★★★

Similar to the comparison of graphs, it is also challenging to identify a universally "best" index. Instead, we summarize the characteristics of different indices in Table 20. We can see that the *relationship index* tends to have a larger size, whereas the *community index* is more compact but incurs the highest construction cost in terms of tokens and time. Indeed, in our experiments, we report the token costs associated with constructing the *community index* for each dataset. Notably, for some datasets, such as HotpotQA and ALCE, this cost is comparable to that of building the graph itself, exceeding 10^8 tokens. Despite their differences, all indices share a common goal of facilitating fast and effective retrieval within graph-based RAG systems. The selection of a specific index should align with the retrieval strategy and the demands of the downstream QA task. For instance, to answer abstract questions that rely on some high-level summaries, it is better to use the community-based indexes (e.g., the index of `GraphRAG`) to retrieve relevant information since they often provide the summary reports for the communities.

B.7 Relevance to the data management community

We agree that graph-based RAG is very relevant to the information retrieval and NLP communities, but we also believe that it is highly relevant to the graph data management community, since it has received growing interest from both industry and academia. In the industrial area, Ant-Group recently proposed Chat2Graph [6], a graph-native agent system that incorporates graph-based RAG as a core component. Similarly, modern graph database systems such as

Prompt for LLM-based multi-dimensional comparison

Prompt:
You will evaluate two answers to the same question based on three criteria: **Comprehensiveness**, **Diversity**, **Empowerment**, and **Directness**.

- Comprehensiveness: How much detail does the answer provide to cover all aspects and details of the question?
- Diversity: How varied and rich is the answer in providing different perspectives and insights on the question?
- Empowerment: How well does the answer help the reader understand and make informed judgments about the topic?
- Directness: How specifically and clearly does the answer address the question?

For each criterion, choose the better answer (either Answer 1 or Answer 2) and explain why. Then, select an overall winner based on these four categories.
Here is the **question**:

```
Question: {query}
```

Here are the two answers:

```
Answer 1: {answer1}
Answer 2: {answer2}
```

Evaluate both answers using the four criteria listed above and provide detailed explanations for each criterion. Output your evaluation in the following JSON format:

```
{
    "Comprehensiveness": {
        "Winner": "[Answer 1 or Answer 2]",
        "Explanation": "[Provide one sentence explanation here]"
    },
    "Diversity": {
        "Winner": "[Answer 1 or Answer 2]",
        "Explanation": "[Provide one sentence explanation here]"
    },
    "Empowerment": {
        "Winner": "[Answer 1 or Answer 2]",
        "Explanation": "[Provide one sentence explanation here]"
    },
    "Overall Winner": {
        "Winner": "[Answer 1 or Answer 2]",
        "Explanation": "[Briefly summarize why this answer is the overall winner]"
    }
}
```

Output:

Figure 35: The prompt for the evaluation of abstract QA.

Neo4j [62], NebulaGraph [60], and PostgreSQL [68] are beginning to support graph-based RAG methods, further reinforcing the growing intersection between AI and graph data management. In the academia area, interest in RAG is also rising within the data management field, as demonstrated by recent work such as `Chameleon` [36] (VLDB 2025) and `Cache-Craft` [2] (SIGMOD 2025).

The core retrieval stage in graph-based RAG systems can be naturally framed as a query optimization process over graph databases, where the goal is to identify the most relevant subgraphs, paths, nodes, or relationships from a large graph G given a natural language query Q. This introduces new challenges for graph query processing and indexing, as many modern graph databases lack native support for the complex retrieval operators required by graph-based RAG workflows. For instance, while retrieving k-hop neighbors is commonly supported, operators such as efficiently identifying communities containing specific entities, computing Steiner trees, or finding k-hop semantically relevant paths to the query Q remain largely unsupported. Moreover, retrieval operators that rely on LLM-based reasoning or semantic similarity are fundamentally beyond the capabilities of existing graph data systems. Our work highlights these limitations and identifies opportunities for extending graph databases to such operators over large graph data—an emerging direction highly relevant to the graph data management community.

Lastly, our work has garnered significant attention from both academic and industrial areas. Our open-source project has attracted

An example abstract question

Question: How do the syntactic and semantic differences between functional and object-oriented paradigms affect program design?

Responses of different methods

Comprehensiveness:

▶ **Why it matters:** *A comprehensive answer should mention key differences and explain their design implications.*

▶ **Comprehensive Answer:** Functional programming emphasizes *pure functions, immutability, and recursion*, reducing side effects and improving testability. Object-oriented programming (OOP) uses *classes, encapsulation, and mutable state*, supporting modularity and reuse. These paradigms shape program design in *syntax* (e.g., *method chaining vs. composition*) and *semantics* (e.g., *stateful vs. stateless logic*). For example, FP promotes *declarative flows* and *better concurrency*, while OOP mirrors *real-world models*. *Understanding both helps in choosing suitable architecture patterns.*

▶ **Incomprehensive Answer:** Functional programming focuses on using *functions*, and object-oriented programming uses *objects and classes*. These differences change how you write code, but *both are just different ways to program*. Most people use OOP because it's more common. Functional programming is used sometimes, but not always necessary. *It depends on what you're doing*, but overall, they just represent two styles.

▶ **Explanation:** *The second response omits key distinctions like mutability and side effects, and lacks detail on how such differences influence program structure or architecture.*

Diversity:

▶ **Why it matters:** *A diverse answer should present multiple angles, use cases, and comparisons.*

▶ **Diverse Answer:** Functional and object-oriented paradigms differ across many dimensions. Syntactically, FP uses *higher-order functions* and *expressions*, while OOP relies on *method calls* and *class hierarchies*. Semantically, FP enforces *immutability* and *stateless computation*, whereas OOP supports *encapsulated state* and side effects. These lead to varied design strategies: FP suits *parallel computation*, *mathematical modeling*, and *data transformation*; OOP works well for *user interfaces*, *simulations*, and *domain modeling*. *Hybrid designs* are also common in modern software.

▶ **Low Diversity Answer:** Functional programming is good for *parallelism* because it doesn't use shared state. Object-oriented programming helps organize code using *classes*. In general, most developers stick to OOP because it's easier and more practical. *Functional programming is more theoretical*. While they work differently, the main thing to know is that *functional is more mathematical*, and object-oriented is more practical for apps.

▶ **Explanation:** *The second answer focuses on only one or two differences and lacks variety in perspectives, missing concrete use cases and deeper design implications.*

Empowerment

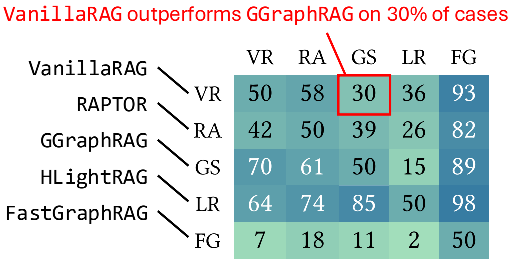
▶ **Why it matters:** *An empowering answer helps readers apply the concepts to make better design decisions.*

▶ **Empowerment Answer:** Understanding the core trade-offs between functional and object-oriented paradigms equips developers to design more robust systems. For instance, choosing *functional principles like immutability* can minimize bugs in *concurrent applications*. OOP's modeling of *real-world entities* simplifies maintenance in large systems. Awareness of these options enables *informed architectural decisions*, such as using a *functional core within an object-oriented shell*. This empowers developers to *select the right paradigm or blend* based on project needs.

▶ **Low Empowerment Answer:** Object-oriented programming is the *standard approach*, so you should usually stick to that. *Functional programming is more academic* and harder to understand. Unless you're working on something very technical, like machine learning or math-heavy software, *it's not really useful*. Most beginners don't need to worry about it. *Just learn OOP and you'll be fine* for most jobs or projects.

▶ **Explanation:** *The second answer discourages understanding by dismissing functional programming as impractical, instead of helping the reader see when and why each paradigm is appropriate.*

Figure 36: Representative good and bad answers across three evaluation dimensions. Cyan italics highlight informative content in good answers; gray italics indicate vague or unhelpful points in bad answers.


VanillaRAG outperforms GGraphRAG on 30% of cases

	VR	RA	GS	LR	FG
VR	50	58	30	36	93
RA	42	50	39	26	82
GS	70	61	50	15	89
LR	64	74	85	50	98
FG	7	18	11	2	50

Figure 37: An example head-to-head comparison result.

significant attention (e.g., 1.3k+ GitHub stars [1]) and has been successfully adopted by *Huawei Cloud for domain-specific QA*, thanks to its modular architecture and ability to handle complex retrieval workflows efficiently. In summary, as noted in the meta-review, we believe that with the increasing importance of graphs and the growing adoption of AI techniques such as RAG, it is both timely and relevant to the graph data management community.

B.8 More examples

• **An example of building graph.** As shown in Figure 38, we consider a corpus from the IT domain, which is segmented into six chunks. We then illustrate how to construct different types of graphs based on this corpus:

- *Tree:* We first apply the Gaussian Mixture clustering algorithm to group the six chunks into two clusters: (chunk 1, chunk 3, chunk 4) and (chunk 2, chunk 5, chunk 6). Each cluster forms a parent node in the second layer, where an LLM is used to summarize the content of all associated chunks. These two parent nodes are then clustered into a single root node (third layer), whose content is also summarized via LLM. The resulting structure is a three-layer tree with 6 leaf nodes (original chunks), 2 intermediate cluster nodes, and 1 root node.
- *Passage Graph:* We construct a passage graph where each chunk is treated as a node, and an edge is added between two nodes if their corresponding chunks share at least τ common entities. For example, with $\tau = 2$, an edge is added between chunk 2 and chunk 4 since they both contain the entities "ChatGPT" and "OpenAI".
- *Knowledge Graph:* We use an LLM to extract entity–relation triples from each chunk. Each entity includes a name, while each relationship is represented by a name and an associated weight indicating how frequently it appears within the given chunk. For example, from chunk 1, we extract the entity "Elon Musk" and its relationship "Founder of".
- *Textual & Rich Knowledge Graph:* The two types of graphs are constructed in a similar way: for each chunk, we use an LLM to extract entities and their relationships. Each entity is represented with three attributes—name, type, and description—while each relationship includes a name, description, and weight. The key difference lies in that the Rich Knowledge Graph further annotates each relationship with a set of keywords, providing more semantic cues for retrieval.

• **Details of operators.** Specifically, we abstract three operators under the **Chunk type**: ❶ Aggregator, ❷ FromRel, and ❸ Occurrence.

- Aggregator: This operator relies on two matrices: a score vector $\mathcal{R} \in \mathbb{R}^{1\times m}$ and an interaction matrix $\mathcal{M} \in \mathbb{R}^{m\times c}$, where m is the number of relationships and c is the number of chunks. Specifically, the i-th entry in $\mathcal{R}$ represents the score of the i-th relationship, while $\mathcal{M}_{i,j} = 1$ if the i-th relationship is extracted from the j-th chunk, and 0 otherwise. Afterwards, the aggregated score of each chunk is computed via matrix multiplication:
$$\Psi = \mathcal{R} \times \mathcal{M},$$
where each entry in $\Psi \in \mathbb{R}^{1\times c}$ represents the aggregated relationship score of a chunk. Based on Ψ, the top-k chunks with the highest scores are selected for retrieval.
- FromRel: For each relationship in the graph, we maintain a mapping to the set of chunks from which it is extracted. This allows us to efficiently retrieve relevant chunks given a set of relationships by computing the union of all chunks associated with these relationships.
- Occurrence: For every relationship, we identify its two associated entities. For each entity, we maintain a mapping to the set of chunks from which it is extracted, since an entity can appear in multiple chunks. If a chunk contains both entities of a given relationship, its score (initially set to 0) is incremented by 1. After processing all relationships, we obtain a score for each chunk, and select the top-k chunks based on these scores.

Besides, there are two operators under the **Community type**: ❶ Entity, and ❷ Layer:

- Entity: This operator retrieves communities that contain the specified entities. Each community maintains a list of associated entities. Retrieved communities are then ranked based on their relevance scores (generated by the LLM), and the top-k communities are returned.
- Layer: In GGraphRAG, communities are detected using the Leiden algorithm, resulting in a hierarchical structure where higher layers represent more abstract and coarse-grained information. The Layer operator retrieves all communities at or below a specified layer, allowing access to more fine-grained community information.

• **An example of Map-Reduce strategy.** In the Map-Reduce phase of GGraphRAG, each retrieved community is individually used to answer the question. Specifically, for each community, the LLM is prompted to generate a partial answer along with a confidence score (ranging from 0 to 100), reflecting how well the community summary addresses the query. After processing all communities, we obtain a set of partial answers, each represented as a pair: (answer, score). These partial answers are then ranked in descending order by score and sequentially appended to the prompt for final answer generation. We also provide an example to illustrate this.

Example 1. *Figure 39 illustrates how the partial answers are generated and used in* GGraphRAG. *Consider an abstract question: "What are the socio-economic impacts of artificial intelligence on the global labor market?" In this example, three communities are retrieved, and* GGraphRAG *generates partial answers from each one. For instance, using the first community, it produces a partial answer discussing the impact of AI on the job landscape, which receives a relevance score of*

[1] https://github.com/JayLZhou/GraphRAG

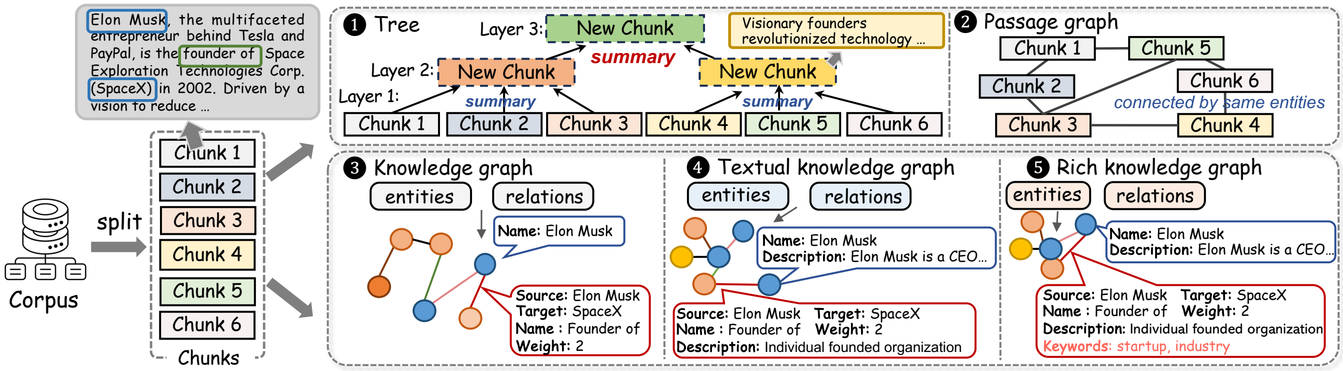
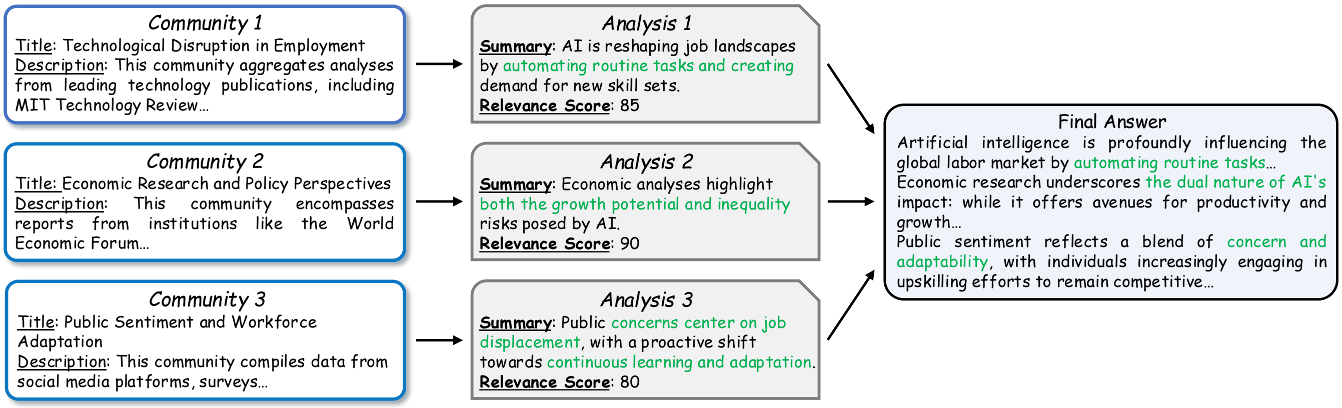


Figure 38: Examples of five types of graphs.



Community 1
Title: Technological Disruption in Employment
Description: This community aggregates analyses from leading technology publications, including MIT Technology Review...

Community 2
Title: Economic Research and Policy Perspectives
Description: This community encompasses reports from institutions like the World Economic Forum...

Community 3
Title: Public Sentiment and Workforce Adaptation
Description: This community compiles data from social media platforms, surveys...

Analysis 1
Summary: AI is reshaping job landscapes by automating routine tasks and creating demand for new skill sets.
Relevance Score: 85

Analysis 2
Summary: Economic analyses highlight both the growth potential and inequality risks posed by AI.
Relevance Score: 90

Analysis 3
Summary: Public concerns center on job displacement, with a proactive shift towards continuous learning and adaptation.
Relevance Score: 80

Final Answer
Artificial intelligence is profoundly influencing the global labor market by automating routine tasks...
Economic research underscores the dual nature of AI's impact: while it offers avenues for productivity and growth...
Public sentiment reflects a blend of concern and adaptability, with individuals increasingly engaging in upskilling efforts to remain competitive...

Figure 39: An illustrative example of GGraphRAG's Map-Reduce generation process. The analysis presents partial answers derived from each community, with the portions incorporated into the final answer highlighted in green.

85. These partial answers are then aggregated, allowing GGraphRAG *to generate a final, comprehensive response.*