GRAG: Graded Retrieval-Augmented Generation for Precise Chunk Retrieval

Tushar Soni *tusharsoni.info@gmail.com*
Independent Researcher

Abstract

Standard RAG systems flatten heterogeneous corpora into a single chunk index, causing *global chunk-pool pollution*: globally plausible but document-incorrect passages pollute the candidate pool—the *document zoo problem*. This is an architectural defect, not an embedding or generator deficiency. **GRAG** (Graded Retrieval-Augmented Generation) resolves it with a three-stage pipeline: (1) document-routing via hybrid scoring over per-document summaries, followed by (2) intra-document hybrid chunk retrieval, deterministic neighbor expansion, and (3) finally reranking the merged candidates via a second hybrid pass. *Graded* refers to progressive narrowing of the retrieval space (corpus → collection → local window), not graded relevance. The pipeline runs fully locally via ONNX; ingestion-time multimodal parsing anchors image descriptions in-place for text-based retrieval. Evaluation on a heterogeneous corpus shows consistent improvements in faithfulness, answer relevance, and context precision over flat-index baselines. The primary bottleneck is structural: fixing *where* retrieval occurs outweighs improving how candidates are scored within a polluted pool.

1 Introduction

RAG grounds language models by conditioning generation on retrieved passages Lewis et al. (2020). On heterogeneous corpora—mixing job descriptions, meeting minutes, schedules, financial records, policies, and research articles—a single flat chunk index fails structurally: chunks from unrelated documents compete in the same ranking space, introducing document-incorrect passages before any local evidence can be assessed (*global chunk-pool pollution*). This is closely related to the broader multi-document RAG challenge Levy et al. (2025); Chen et al. (2024). Even when a partially relevant chunk is recovered, adjacent rows, clauses, or continuation spans needed to complete a table or schedule entry are often absent.

Two unsatisfactory extremes emerge. *Long-context*: large retrieved windows may contain the correct fact, but dilution causes "lost in the middle" failures Liu et al. (2024)—the generator attends to the wrong neighborhood. *Short-context*: small retrieved sets are efficient but highly sensitive to ranking errors; document-incorrect candidates from the flat pool corrupt the evidence package. Neither extreme solves the document zoo problem.

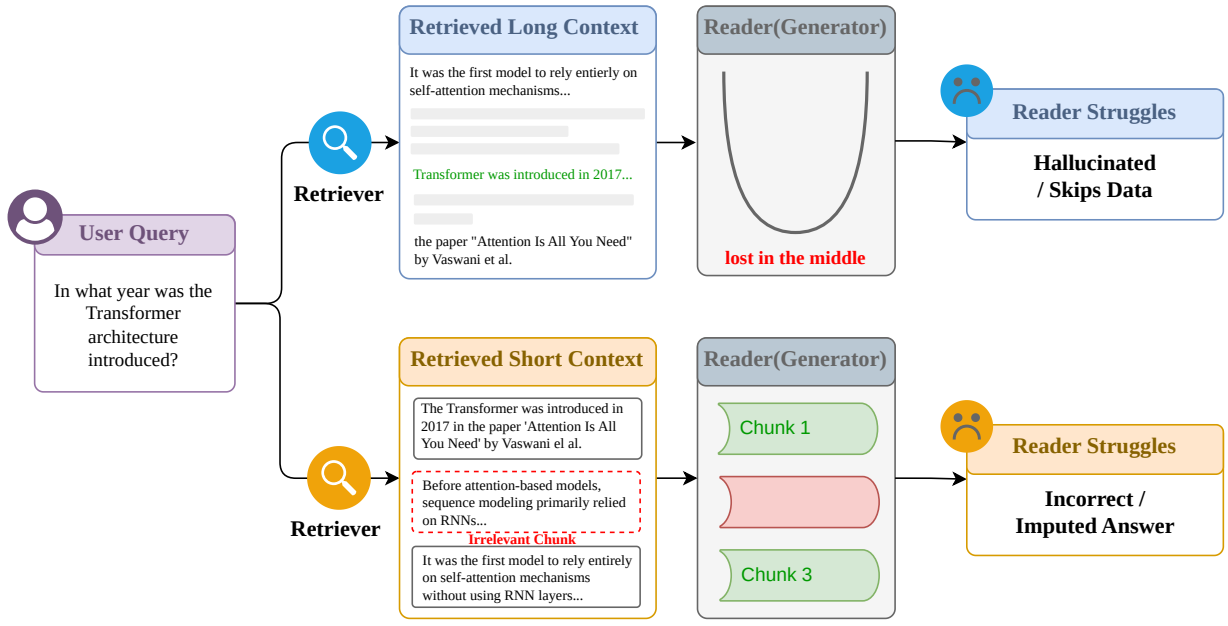

Figure 1: **Failure modes of conventional RAG under long-context and short-context evidence regimes.** The same query is handled by two retrieval strategies. *Top:* a long retrieved context may bury the answer-bearing span among surrounding text, so the reader exhibits a "lost in the middle" pattern and hallucinates or skips critical facts. *Bottom:* a short retrieved set can surface a compact answer fragment but also admit irrelevant chunks from competing sources, so the reader produces an incorrect or imputed answer.

GRAG addresses this by decomposing retrieval into progressive stages: document routing → local hybrid retrieval → neighbor expansion → reranking. The reference implementation is **ManuIndex**, a local ONNX-oriented system. Limitations are in Section 6.3.

2 Background and Related Work

2.1 Flat Pipelines: Dense RAG, Hybrid RAG, and Query Rewrite

Dense RAG Lewis et al. (2020) splits documents into fixed-size overlapping chunks, embeds them in a single FAISS index, and ranks by vector similarity. *Hybrid RAG* Sawarkar et al. (2024) fuses dense similarity with BM25 over the same flat pool. *Query-rewrite RAG* Ma et al. (2023) generates retrieval-oriented query rewrites before issuing them against the flat index, improving recall at the cost of extra tokens and latency.

All three share the same structural premise: chunk ranking over a global pool without document-level gating. Hybrid fusion can promote better-matching fragments from the *wrong* source; query rewriting surfaces more paraphrases of already-polluted candidates Ma et al. (2023). GRAG reuses hybrid dense–sparse search *after* routing, treating these as useful local operators rather than architectural solutions.

2.2 LongRAG: Long Retrieval Units and MaxP Aggregation

A complementary line of work attacks the short-context incompleteness illustrated in Figure 1 by enlarging the retrieval unit rather than refining chunk ranking alone. LongRAG-style pipelines group fine-grained MaxP-style subchunks into long retrieval units on the order of several thousand tokens, score those units by aggregating subchunk similarities (commonly MaxP: the unit score is the best constituent subchunk score), and return the top long units to a long-context reader Jiang et al. (2024). In the baseline realization used

here, consecutive fixed-size subchunks are packed until a unit-size budget is reached; for short documents the unit collapses to the full document text.

This design improves local continuity: tables, multi-sentence definitions, and adjacent clauses are more likely to travel together than under pure short-chunk RAG. It does not, however, resolve the document-zoo problem, and it reopens the long-context failure mode of Figure 1. Once a long unit is selected, the generator must attend over a wide window in which the answer span may sit far from the prompt edges ("lost in the middle"), raising the risk of hallucination or skipped facts even when the correct evidence is present. When multiple long units from different sources enter the context, pollution and mid-prompt underuse can co-occur. GRAG takes a different stance on completeness: after document-level routing narrows the pool, it reconstructs only a *short* local neighborhood around each seed chunk rather than promoting multi-kilotoken units as the primary retrieval object.

2.3 Hierarchical RAG (H-RAG)

H-RAG et al. (2026) retrieves fine-grained child windows and maps selected children back to larger parent spans for generation, improving local completeness over isolated short chunks. H-RAG and GRAG both reject flat global competition, but on different axes: H-RAG's hierarchy is *intra-document* (child $\rightarrow$ parent span), while GRAG's primary gate is *inter-document* (query $\rightarrow$ top-c collections). Parent-child hierarchy alone does not prevent high-scoring children from the wrong document from promoting the wrong parent. GRAG applies local hybrid retrieval and neighbor expansion only *inside* routed collections.

3 GRAG Architecture

3.1 Problem Setting

Let corpus $D = \{d_1, d_2, \ldots, d_n\}$. Flat RAG inserts all chunks into one global index; on heterogeneous corpora, semantically plausible but document-incorrect chunks pollute retrieval. GRAG restructures retrieval into three levels:

1. Document-level routing, which identifies the top candidate document collections for a query.
2. Local chunk-level retrieval, which searches only within those selected collections.
3. Deterministic neighbor expansion, which recovers adjacent chunks to improve local completeness.

Many queries are document-selective before they are chunk-selective; routing resolves document identity first, so chunk ranking no longer competes across the full zoo.

3.2 Three-Stage Retrieval Pipeline

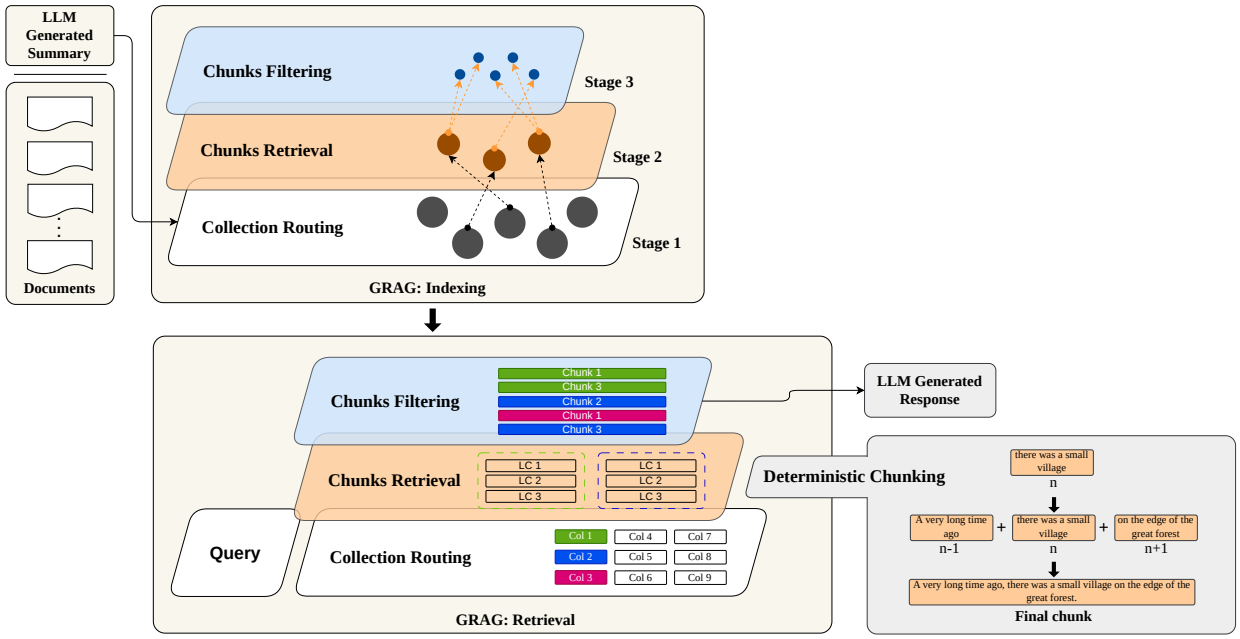

Figure 2: **GRAG retrieval architecture.** A query is first matched against document-summary metadata to identify the most relevant collections. Retrieval is then restricted to those routed collections, after which local chunk retrieval, neighbor expansion, and reranking produce the final evidence set for generation.

Ingestion (per document): generate a compact routing summary, embed it, store as metadata; split the document into deterministic non-overlapping chunks; build a per-document dense and sparse index.

Query time (Algorithm A3):

1. Embed the query; compare against all summary embeddings; select top-c collections.
2. For each collection: run MMR dense + BM25 sparse, then fuse into hybrid candidates.
3. Expand each candidate with immediate neighbors.
4. Deduplicate, rerank the merged pool, and return top-k.

Complexity. Routing: $O(|D|)$. Per-collection retrieval: $O(|C_d| \log |C_d|)$ FAISS + $O(|C_d| \cdot L)$ BM25. Total: $O(|D| + c \cdot |C_d| \log |C_d|)$ versus $O(|D| \cdot |C_d| \log(|D| \cdot |C_d|))$ for flat search. When $c \ll |D|$, search-space reduction is substantial. Storage scales as $O(|D| \cdot |C_d|)$—asymptotically identical to flat RAG, with per-document index management overhead.

3.3 Document-Level Routing

Each document is summarized (LLM over Markdown headings if representative, else full document; 4–5 dense sentences). The summary is embedded and stored with the document ID. At query time, routing score is:

$$\text{score}_{\text{route}}(d_j \mid q) = \hat{\mathbf{q}} \cdot \hat{\mathbf{s}}_j$$

where $\hat{\mathbf{q}}$ and $\hat{\mathbf{s}}_j$ are normalized embeddings. The routed set is:

$$\mathcal{D}_c(q) = \mathrm{TopC}_{d_j \in D} \ \mathrm{score}_{\mathrm{route}}(d_j \mid q)$$

3.4 Local Hybrid Retrieval

Within each routed collection, dense retrieval uses MMR Carbonell and Goldstein (1998):

$$x^* = \arg\max_{x_i \in \mathcal{X} \setminus \mathcal{R}} \left[\lambda \cdot \cos(\mathbf{x}_i, \mathbf{q}) - (1 - \lambda) \cdot \max_{x_j \in \mathcal{R}} \cos(\mathbf{x}_i, \mathbf{x}_j) \right]$$

Sparse retrieval uses BM25. The hybrid score is:

$$\mathrm{score}_{\mathrm{hybrid}}(x) = \alpha \ \mathrm{score}_{\mathrm{dense}}(x) + (1 - \alpha) \ \mathrm{score}_{\mathrm{sparse}}(x)$$

3.5 Neighbor Chunk Expansion

Each retrieved seed chunk at index k is expanded to a radius-1 window:

$$N(k) = \{k - 1, \ k, \ k + 1\} \cap \mathrm{dom}(M)$$

Non-overlapping usable window (given already-emitted set U):

$$W(k, U) = \{i \in N(k) \mid i \notin U\}$$

This recovers adjacent rows, clauses, or continuation spans for tables and semi-structured content without unbounded context growth.

3.6 Reranking

After routing, local hybrid retrieval, and neighbor expansion, a second hybrid pass re-evaluates the merged, deduplicated candidate pool from all routed collections. Candidates are re-embedded for the dense component, ensuring stable global ranking after neighborhood expansion. This returns the final top-k contexts for generation.

4 High-Fidelity Parsing for Heterogeneous Documents

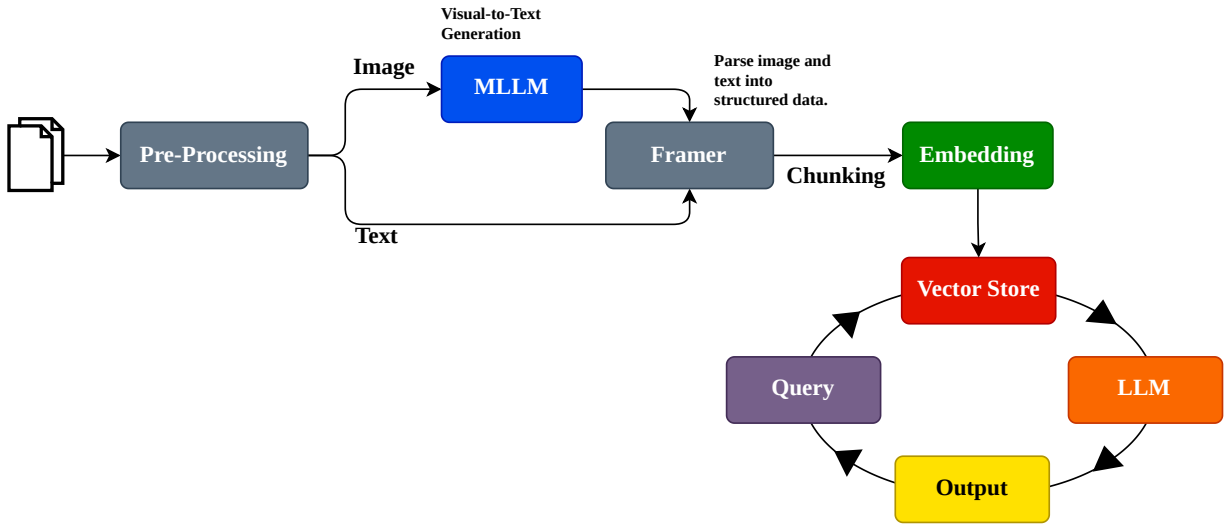

Figure 3: **Vision-aware document parsing workflow.** Images are processed once at ingestion with an MLLM; descriptions are reinserted in place at their original image locations; downstream retrieval is fully textual.

For a document $d = [t_1, i_1, t_2, i_2, \ldots, t_m]$ with MLLM description $g(i_r)$ per image, the parsed document is:

$$\tilde{d} = [t_1, g(i_1), t_2, g(i_2), \ldots, t_m]$$

Structural reinsertion preserves positional semantics: image-derived evidence stays co-located with the prose that references it. This converts tables, figures, and non-prose layouts from retrieval dead zones into searchable text while moving multimodal inference cost from query time to ingestion time—consistent with recent multimodal RAG work Tripathi et al. (2025); Tanaka et al. (2025); Cho et al. (2024); Yu et al. (2024).

5 Experiments

5.1 Setup

Datasets: `iam-tsr/ragmix` (heterogeneous: job descriptions, meeting minutes, financial summaries, policy docs) and `neural-bridge/rag-dataset-12000`. Results are reported separately to avoid masking task-specific differences.

Retrieval budget: $k = 3$ for all methods. Chunk sizes: flat methods use 150-token chunks; H-RAG uses 512-token parents; LongRAG uses roughly 4,012-token units.

Embeddings: BERT ONNX and Qwen3-Embedding 0.6B ONNX, with the same configuration applied across all methods per panel.

Generator: Qwen2.5-3B.

Metrics (all @3): Faithfulness, Answer Recall, Answer F1, Context F1, Time (s/question), total tokens/question. A supplementary **routing recall** metric tracks the fraction of queries for which the top-c routing step includes the ground-truth document, which is a direct diagnostic for the first-stage bottleneck.

Note on BERT results: GRAG's routing is a hard narrowing gate; weaker query-to-summary alignment (BERT vs. Qwen3-Embedding) can exclude the correct collection before local stages run. This explains the

pattern of lower GRAG scores under BERT and is architectural, not a general claim about BERT for flat retrieval.

5.2 Results

Five baselines span the design space: Naive RAG (flat dense index), Flat Hybrid RAG (MMR + BM25, flat index), Query Rewrite RAG (LLM rewrite → flat dense), H-RAG (child retrieval → parent mapping), and LongRAG (4k-token units, MaxP scoring).

Table 1: Baseline evaluation on `iam-tsr/ragmix`. **Bold** = best; underline = second-best.

Embedding Model	Method	Quality		Answer		Efficiency	
		Faith. ↑	Ctx F1@3 ↑	AR@3 ↑	F1@3 ↑	Time (s) ↓	Tokens ↓
BERT (ONNX)	Naive	0.3612	0.2641	0.2559	0.2300	2.659	**297.0**
	Flat Hybrid	0.3747	0.3061	0.2753	0.2106	2.758	347.3
	Query Rewrite	0.3612	0.2641	0.2559	0.2300	4.021	615.0
	Hierarchical	0.4964	**0.3722**	0.3767	**0.2992**	3.251	979.6
	LongRAG	0.5252	0.3609	**0.3786**	0.2979	**2.300**	3143.5
	GRAG	**0.5257**	0.3018	0.2822	0.2294	2.826	476.6
Qwen3 Embedding 0.6B (ONNX)	Naive	0.3864	0.4114	0.3649	0.2812	**2.414**	398.7
	Flat Hybrid	0.4831	0.4366	0.3691	0.2846	2.456	**395.5**
	Query Rewrite	0.3864	0.4114	0.3649	0.2812	3.646	716.8
	Hierarchical	0.5876	0.5004	0.4304	0.3512	2.654	972.0
	LongRAG	0.6272	**0.5792**	**0.5112**	0.3685	2.606	3040.8
	GRAG	**0.7058**	0.5440	0.5097	**0.4034**	2.516	562.8

Table 2: Evaluation on `neural-bridge/rag-dataset-12000`. **Bold** = best; underline = second-best.

Embedding Model	Method	Quality		Answer		Efficiency	
		Faith. ↑	Ctx F1@3 ↑	AR@3 ↑	F1@3 ↑	Time (s) ↓	Tokens ↓
BERT (ONNX)	Naive	0.7302	0.5663	0.5272	0.2584	2.383	584.9
	Flat Hybrid	0.7144	0.5934	0.5339	0.2676	2.460	599.2
	Query Rewrite	0.7302	0.5663	0.5272	0.2584	3.481	860.9
	Hierarchical	0.7744	**0.6900**	**0.5961**	0.3431	2.062	795.2
	LongRAG	0.7716	0.6427	0.5937	**0.3622**	**1.804**	3114.8
	GRAG	**0.7755**	0.5938	0.4429	0.3462	2.521	**531.5**
Qwen3 Embedding 0.6B (ONNX)	Naive	0.7761	0.7189	0.5886	0.3125	2.339	**582.6**
	Flat Hybrid	0.7448	0.7073	0.5981	0.2979	2.596	629.0
	Query Rewrite	0.7761	0.7189	0.5886	0.3125	3.454	858.6
	Hierarchical	0.8893	**0.8366**	0.6650	0.3840	2.049	918.3
	LongRAG	0.8437	0.8256	0.6449	**0.3850**	**1.979**	2979.6
	GRAG	**0.9257**	0.8074	**0.6901**	0.3786	2.660	645.4

Analysis. GRAG ranks first in Faithfulness across all four dataset–embedding panels. The margin is decisive under Qwen3-Embedding (+12.5% over LongRAG on RAGMix; +4.1% over H-RAG on Neural-Bridge) but near-zero under BERT (+0.1% in both). This divergence directly reflects routing sensitivity: BERT's weaker query-to-summary alignment increases the risk of routing the correct collection out of the top-c

c set before local retrieval begins—a hard, unrecoverable loss. Switching to Qwen3 raises GRAG's RAGMix faithfulness by +34.3% (0.5257 → 0.7058), far outpacing the same switch for any baseline, confirming that the first-stage routing gate amplifies embedding quality differences more than flat-index methods do.

The Context F1 trade-off is real and expected. H-RAG and LongRAG exceed GRAG on Context F1 in three of four panels (gaps of -0.03 to -0.10), because parent-span mapping and multi-kilotoken units maximise raw context coverage. Yet that coverage does not translate into higher faithfulness—GRAG outperforms both on generation grounding despite returning less total context. The implication is that retrieval precision within the correct document contributes more to answer faithfulness than maximising the volume of retrieved text.

Query Rewrite produces faithfulness identical to Naive RAG ($\Delta = 0.000$ in all four panels) at 1.5× the latency and roughly 2× the token cost. Improving the query representation does not help when the underlying candidate pool remains globally polluted. LongRAG achieves strong context coverage and Answer Recall but at $4.6 - 6.6\times$ GRAG's token cost; GRAG delivers competitive or superior Answer F1 at a fraction of that budget (562-645 tokens vs. 2,980-3,144).

6 Discussion

6.1 Mechanistic Insights

GRAG's benefit is structural: it changes where retrieval occurs, not just how candidates are scored. Routing reduces global pollution before chunk ranking begins; neighbor expansion restores local evidence missed by single-chunk retrieval; reranking stabilizes the final ordering. The combination is most effective when exact answers depend on nearby rows, clauses, dates, or role-specific fields.

6.2 Efficiency and Resource Trade-offs

GRAG shifts computational cost from query time to ingestion time: routing vectors are built once; query-time retrieval is bounded to c collections; neighbor expansion is $O(k)$; multimodal parsing is performed once per document. This keeps query-time inference lean and fully local.

6.3 Core Limitations and Scope of Validity

Routing failure. Routing is a hard gate—a missed collection is unrecoverable downstream. Quality depends on LLM-generated summaries; dense jargon, non-standard formatting, or heavy tables may yield poor routing representations. Future work: soft routing with entropy-based confidence, adaptive c, routing-aware embedding fine-tuning.

Storage and expansion radius. One dense + one sparse index per document introduces per-document management overhead that becomes non-trivial at tens of thousands of documents. The fixed ± 1 expansion radius may be too narrow for large tables or multi-clause definitions; an adaptive radius would improve coverage.

Cross-document queries. Multi-hop queries requiring evidence from multiple documents may be underserved when c is small. Ingestion-time MLLM descriptions are a lossy compression; complex charts may lose quantitative detail not expressible in brief text.

6.4 Parsing as an Enabling Factor

Better structure preservation improves all downstream retrieval systems. In-place reinsertion of image descriptions changes the retrieval problem: instead of searching a large cross-document visual pool, the retriever

finds image-derived evidence anchored to the text that references it. GRAG's contribution combines high-fidelity document preparation with document-aware retrieval structure—the two are not independent.

7 Conclusion

GRAG addresses the *document zoo problem* architecturally: it routes queries to a bounded set of candidate collections, applies local hybrid retrieval, deterministic neighbor expansion, and a final reranking pass, while shifting multimodal parsing cost to ingestion time. Evaluation confirms highest faithfulness across all panels and best answer quality on the heterogeneous RAGMix corpus. Routing is the primary performance-sensitive component: embedding misalignment can exclude the correct collection before local stages run, motivating future work on adaptive routing, multi-document query support, and routing-aware fine-tuning. The central finding is that improving RAG on heterogeneous corpora requires structural correction to *where* retrieval occurs.

References

Jaime G. Carbonell and Jade Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proc. 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '98)*, pages 335–336, 1998. doi: 10.1145/290941.291025.

Xuanhe Chen, Peitian Gao, Jiafeng Song, and Xiaohui Tan. HiQA: A hierarchical contextual augmentation RAG for multi-documents QA. *arXiv preprint arXiv:2402.01767*, 2024.

J. Cho, D. Mahata, O. İrsoy, Y. He, and M. Bansal. M3DocRAG: Multi-modal retrieval is what you need for multi-page multi-document understanding. *arXiv preprint arXiv:2411.04952*, 2024.

P. Elchafei et al. H-RAG at SemEval-2026 task 8: Hierarchical parent-child retrieval for multi-turn RAG conversations. In *Proc. SemEval-2026*, 2026. URL https://arxiv.org/abs/2605.00631.

Z. Jiang, X. Ma, and W. Chen. LongRAG: Enhancing retrieval-augmented generation with long-context LLMs. *arXiv preprint arXiv:2406.15319*, 2024.

Shahar Levy, Noam Mazor, Liel Shalmon, Michael Hassid, and Gabriel Stanovsky. More documents, same length: Isolating the challenge of multiple documents in RAG. *arXiv preprint arXiv:2503.04388*, 2025.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL https://arxiv.org/abs/2005.11401.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638.

X. Ma, Y. Gong, P. He, H. Zhao, and N. Duan. Query rewriting for retrieval-augmented large language models. *arXiv preprint arXiv:2305.14283*, 2023.

K. Sawarkar, A. Mangal, and S. R. Solanki. Blended RAG: Improving RAG accuracy with semantic search and hybrid query-based retrievers. In *Proc. IEEE International Conference on Multimedia Information Processing and Retrieval (MIPR)*, 2024. URL https://arxiv.org/abs/2404.07220.

R. Tanaka, T. Iki, T. Hasegawa, K. Nishida, K. Saito, and J. Suzuki. VDocRAG: Retrieval-augmented generation over visually-rich documents. *arXiv preprint arXiv:2504.09795*, 2025.

V. Tripathi, T. Odapally, I. Das, U. Allu, and B. Ahmed. Vision-guided chunking is all you need: Enhancing RAG with multimodal document understanding. *arXiv preprint arXiv:2506.16035*, 2025.

S. Yu, C. Tang, B. Xu, J. Cui, J. Ran, Y. Yan, Z. Liu, S. Wang, X. Han, Z. Liu, and M. Sun. VisRAG: Vision-based retrieval-augmented generation on multi-modality documents. *arXiv preprint arXiv:2410.10594*, 2024.

A Detailed Algorithm of GRAG

Algorithm 1 Document Ingestion and Routing Summary Construction

Require: Document D (raw text), chunk size s
Ensure: Persisted dense index V_d, sparse index V_s, routing entry $(d, \mathbf{e}, \sigma)$

$d \leftarrow \text{GenerateUniqueID}()$ ▷ document identifier
$\sigma \leftarrow \text{Summarize}(D)$ ▷ LLM-generated document summary
$\mathbf{e} \leftarrow \text{Embed}(\sigma)$ ▷ summary embedding vector
$M \leftarrow \text{LoadMetadata}() \cup \{(d, \mathbf{e}, \sigma)\}$
$\text{PersistMetadata}(M)$ ▷ routing table update
$C \leftarrow \text{RecursiveSplit}(D, s)$ ▷ deterministic, non-overlapping chunks
for $i \leftarrow 0$ to $|C| - 1$ **do**
    $C_i.\text{index} \leftarrow i$ ▷ tag chunk with positional index
end for
$V_s \leftarrow \text{BuildBM25}(C)$; $\text{Persist}(V_s, d)$
$V_d \leftarrow \text{BuildFAISS}(C, \text{Embed})$; $\text{Persist}(V_d, d)$
return V_d

Algorithm 2 Deterministic Neighbor Chunk Expansion

Require: Retrieved chunks $\hat{C} = \{c_1, \ldots, c_m\}$, indexed chunk map $\Phi : i \mapsto \text{chunk}_i$
Ensure: Expanded, non-overlapping chunk windows $\mathcal{E}$

$\mathcal{E} \leftarrow \emptyset$; $U \leftarrow \emptyset$ ▷ U: set of already-emitted indices
for each $c \in \hat{C}$ **do**
    $i \leftarrow c.\text{index}$
    if i undefined **then**
        $\mathcal{E} \leftarrow \mathcal{E} \cup \{c\}$
        continue
    end if
    if $i \in U$ **then**
        continue ▷ skip already-covered chunk
    end if
    $N \leftarrow \{\, j \in \{i-1, i, i+1\} \;:\; j \in \text{dom}(\Phi) \wedge j \notin U \,\}$
    if $N = \emptyset$ **then**
        continue
    end if
    $\text{window} \leftarrow \text{Concat}\big(\Phi(j) \text{ for } j \in \text{sorted}(N)\big)$
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{\text{window}\}$
    $U \leftarrow U \cup N$ ▷ mark neighbors as consumed to prevent overlap
end for
return $\mathcal{E}$

Algorithm 3 Query-Time GRAG Retrieval

Require: Query q, top collections c, top passages k, MMR trade-off λ, hybrid weight α
Ensure: Ranked list of k passages

$\mathbf{q} \leftarrow \text{Embed}(q)$
$M \leftarrow \text{LoadMetadata}()$
$S \leftarrow \{\, \mathbf{q} \cdot \mathbf{e}_d \;:\; (d, \mathbf{e}_d, \sigma_d) \in M \,\}$ ▷ summary-level similarity
$\mathcal{D} \leftarrow$ top-c document IDs by S ▷ coarse routing over summaries
$\mathcal{X} \leftarrow \emptyset$ ▷ candidate pool
for each $d \in \mathcal{D}$ **do**
    $V_d \leftarrow \text{LoadDenseIndex}(d);\;\; V_s \leftarrow \text{LoadSparseIndex}(d)$
    $R_{\text{dense}} \leftarrow \text{MMRRetriever}(V_d, \lambda)$
    $R_{\text{sparse}} \leftarrow \text{BM25Retriever}(V_s)$
    $R_{\text{hyb}} \leftarrow \text{EnsembleRetriever}(R_{\text{dense}}, R_{\text{sparse}};\; \alpha,\; 1-\alpha)$
    $\hat{C} \leftarrow R_{\text{hyb}}.\text{Invoke}(q)$ ▷ fine-grained retrieval within document
    $\mathcal{X} \leftarrow \mathcal{X} \cup \text{NeighborExpansion}(\hat{C}, V_d)$
end for
$T \leftarrow \text{DeduplicateByContent}(\mathcal{X})$
if $T = \emptyset$ **then**
    return $\emptyset$
end if
$R'_{\text{hyb}} \leftarrow \text{EnsembleRetriever}(\text{MMR}(T, \lambda), \text{BM25}(T);\; \alpha,\; 1-\alpha)$
return top-k of $R'_{\text{hyb}}.\text{Invoke}(q)$