GraphRAG-V: Fast Multi-Hop Retrieval via Text-Chunk Communities

Tengkai Yu[1], Venkatesh Srinivasan[2], and Alex Thomo[1]

[1] University of Victoria, Victoria, BC V8W 3P6, Canada
yutengkai@uvic.ca, thomo@uvic.ca
[2] Santa Clara University, Santa Clara, CA 95053, USA
vsrinivasan4@scu.edu

Abstract. Retrieval-augmented generation (RAG) is the standard way to equip large-language models (LLMs) with knowledge that lies outside their training data. The classic *vector-store* recipe works well when a single, tightly focused passage suffices, but it struggles whenever a question demands evidence scattered across many documents. Microsoft's GraphRAG addresses this by prompting an LLM to extract entity-relation triples from each chunk, then querying a community-partitioned knowledge graph. However, this approach requires expensive LLM calls per chunk, generates thousands of triples, and extends preprocessing time to several hours even for modest-sized corpora.
We show that constructing a triple-based knowledge graph can be skipped entirely. Instead, our proposed method, GraphRAG-V, treats each raw chunk as a node, embeds them once, builds a similarity graph, and applies the scalable VLouvain algorithm to discover chunk-level vector communities in embedding space. A single pass then summarizes each community, enabling the LLM to receive both local and global evidence in one prompt. On the 2,556-question MultiHopRAG benchmark, GraphRAG-V indexes the corpus in minutes, answers all questions in under an hour on a single A100 GPU, and improves recall by eleven percentage points over a strong vector-store baseline. It also outperforms Microsoft's GraphRAG across all metrics, while being orders of magnitude faster. These results suggest that implicit chunk-level structures, not explicit triple-based graphs, are the key to scalable, cost-efficient long-context reasoning.

Keywords: Retrieval-Augmented Generation, Community Detection, GraphRAG, Large-Language Models

1 Introduction

Large language models (LLMs) such as GPT-4 [6, 1], Qwen 2.5 [32, 27], and the Llama 3.1 family [9] achieve impressive few-shot performance, yet they remain limited to knowledge seen during pretraining. Retrieval-augmented generation (RAG) bridges this gap by fetching relevant documents at inference time and adding them to the prompt [15]. The standard *vector-store* approach embeds overlapping text chunks, indexes

them with approximate nearest-neighbor (ANN) search, and prepends the k most similar chunks to the LLM input [14]. While fast and effective for single-hop "needle" queries, this method falters when questions require aggregating evidence from multiple documents: ANN often retrieves redundant passages, misses complementary facts, and exhausts the prompt budget without providing sufficient global context [16, 2].

Microsoft's *GraphRAG* extends the retrieval scope by prompting an LLM to extract entity-relation triples from each chunk, constructing a knowledge graph, applying community detection [28], and retrieving both local neighbors and community summaries at inference time [8]. Though effective, this pipeline is resource-intensive: it issues expensive entity extraction LLM prompts, generates gigabytes of intermediate JSON, and requires hours to build relationship edges even for modest corpora. Failures are common when LLM outputs deviate from strict JSON schemas.

Several successors aim to lighten this load. ArchRAG adds hierarchical structure [29]; HippoRAG applies personalized PageRank [11]; LightRAG introduces dual-level keyword nodes [10]; KET-RAG retains only a skeleton graph plus a keyword-chunk bipartite layer [13]; Think-on-Graph 2.0 alternates between KG traversal and text retrieval [17]; GNN-RAG applies graph neural networks to a pruned chunk graph [19]. Despite their differences, all of these methods still depend on explicit entity extraction and knowledge graph construction, typically pruning edges via k-nearest-neighbor heuristics that risk discarding crucial cross-cluster links.

A separate line of work avoids graphs entirely by clustering chunk embeddings. CRAG uses k-means and retrieves one passage per cluster centroid [2]. While this reduces redundancy, hard clustering optimizes only centroid distance, not the full pairwise similarity between chunks needed for multi-hop reasoning.

We propose GraphRAG-V, a graph-free alternative that retains the benefits of community-based reasoning without relying on explicit entity graphs. Every chunk is embedded once and treated as a node vector; we then apply the GPU-efficient VLouvain algorithm [3], which operates directly on the embedding matrix to optimize modularity over the full cosine similarity graph, without explicitly constructing or storing that graph. Notably, VLouvain achieves the same community structure as classic Louvain [5], but with drastically reduced memory and compute overhead.

Our design highlights a key insight: GraphRAG's gains stem not from explicit Knowledge Graph triples but from exposing the LLM to community structure among text chunks. Furthermore, VLouvain captures all pairwise similarities without kNN pruning, preserving cross-chunks links vital for reasoning. At inference time, we mirror GraphRAG's two-level logic: we return both neighbourhood passages and community summaries using a single ANN lookup, but eliminate all entity extraction, edge materialization, and intermediate storage. Preprocessing time drops from hours to minutes, with no loss in accuracy.

Our contributions are summarized as follows.

- We introduce GraphRAG-V, a graph-free retrieval method that supplies both local and global evidence without constructing a knowledge graph or issuing per-chunk extraction prompts.
- We detect communities directly in chunk embedding space, capturing all chunk pairwise similarities and avoiding the information loss caused by KNN pruning.
- Community detection and summarization are implemented as batched matrix operations, completing in seconds on a single A100 GPU.
- We keep memory usage minimal by storing only the original chunk vector index and compact community summaries, with no need for edge lists, JSON triples, or intermediate graph structures.

2 Related Work

Vector-store RAG. Dense-retrieval pipelines such as DPR and the original RAG framework embed overlapping chunks, store them in an approximate-nearest-neighbour (ANN) index, and prepend the k closest chunks to the query [15, 14]. In-context re-rankers [21], black-box adapters [24], self-reflective loops [4], and the latency-oriented LightRAG variants [10] refine this recipe, typically using Sentence-BERT, MPNet or the BGE family of encoders [22, 25, 7]. Despite these improvements, vector retrieval excels mainly at single-hop "needle" facts. Once an answer requires evidence scattered across documents, nearest- neighbour pruning can discard useful but weaker links, producing redundant passages and leaving multi-hop links unresolved [16, 2].

Graph-based RAG. GraphRAG extracts entity-relation triples with an LLM, stores them as JSON, partitions the resulting knowledge graph, and retrieves both local neighbours and community summaries [8]. The design improves answer depth but issues entity and relation extraction prompts for each chunk and spends hours on edge materialisation even for mid-size corpora. Follow-up work tries to gain efficiency in different ways: HippoRAG adds personalised PageRank [11]; Think-on-Graph alternates KG traversal and text search [17]; GNN-RAG embeds the chunk graph with a GNN [19]; ArchRAG attaches text attributes to a two-level community hierarchy [29]; and the recent KET-RAG framework keeps only a small "skeleton" KG and a keyword-chunk bipartite graph to cut extraction cost [13]. Most of these pipelines still rely on k-nearest-neighbour (KNN) pruning to limit graph size; pruned edges save memory but risk removing exactly the cross-cluster links needed for multi-hop reasoning.

Community detection without explicit graphs. Louvain [5] and its Leiden successor maximise intra-community connectivity and are therefore attractive for grouping semantically related chunks. Their drawback is the need for an explicit edge list. VLouvain shows that when edge weights are inner-product similarities, Louvain's modularity can be optimised directly in embedding space, eliminating the edge list while preserving the objective [3]. Our work builds on VLouvain, treating every chunk as a node and discovering *vector communities* without constructing a knowledge graph, thus avoiding both KNN pruning and triple extraction.

Embedding and indexing foundations. High-quality semantic vectors originate with Sentence-BERT [22] and MPNet [25]; ColBERT-v2 adds late interaction for fine-grained matching [23]. We adopt the multilingual BGE encoder family [7, 31, 33, 30] and store embeddings in an HNSW index [18] for sub-second retrieval.

Evaluation practice. Recent surveys emphasise factual faithfulness and concise evidence selection in RAG evaluation [34, 12]. We follow this guidance, reporting answer-level accuracy and recall from the MultiHopRAG scorer [26] alongside runtime statistics.

3 Methodology

GraphRAG-V grounds an LLM in a hierarchy of *node vector communities*, which are densely connected groups of text chunks obtained *without* an explicit materialization of the chunk similarity graph. During indexing, every chunk is embedded once, VLouvain discovers communities directly in the embedding matrix, and an asynchronous map-reduce prompt distils a short summary for each community. At inference time a query is embedded once and routed through both *local* (fine-grained) and *global* (coarse-grained) retrieval; the resulting evidence is passed to the generator. The entire pipeline stores only vectors and a few kilobytes of JSON summaries: no edges, no entity triples, and no k-nearest pruning.

3.1 VLouvain Overview

VLouvain is a fast, GPU-efficient algorithm that identifies communities (groups of closely related nodes) directly in embedding space, without explicitly constructing a similarity graph [3]. It builds on the classic Louvain method for community detection [5], which traditionally operates on graphs with explicitly defined edges. Instead of requiring a graph, VLouvain treats each node as a vector and measures similarity using cosine similarity. This defines an implicit, fully connected similarity graph, but the graph is never materialized or stored.

The algorithm groups nodes by maximizing modularity [20], a score that captures how well a set of nodes forms a dense, self-contained cluster. For each node, VLouvain checks whether switching to a different community increases this score. These decisions are made using efficient matrix operations, allowing the algorithm to fully exploit GPU parallelism.

The process runs in multiple rounds (typically just two or three), gradually building a hierarchy of communities: small, specific groups at lower levels, and broader, more general ones above. Each round reuses and refines the structure from the previous one. Throughout, no actual graph or edge list is created. The algorithm operates entirely on the embedding matrix, with only compact auxiliary statistics in memory. This makes VLouvain highly scalable, memory-efficient, and practical even for large collections of nodes.

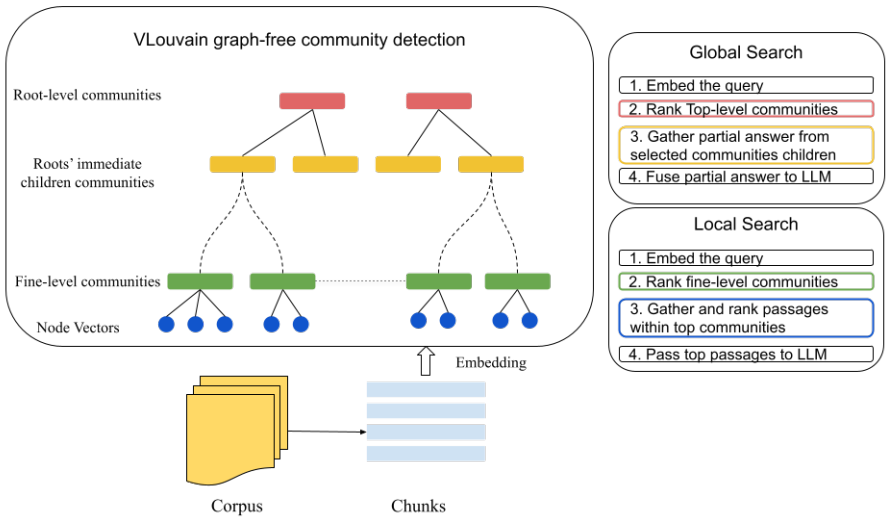

Fig. 1: End-to-end **GraphRAG-V** architecture. *Indexing* (left) embeds chunks, discovers vector communities with VLouvain *without constructing an explicit graph*, and stores only the vectors, community labels, and summaries. *Retrieval* (right) embeds the query once and runs **local** (fine-level passages) and **global** (root-level summaries → child summaries → fusion) pipelines.

3.2 Indexing and Retrieval

Indexing. The corpus is segmented into 600-token chunks with 100-token overlap and encoded by the BGE-M3 encoder, producing unit vectors. Those vectors are inserted into a FAISS HNSW index and, in parallel, augmented and processed by VLouvain on a single GPU, yielding a multi-level chunk hierarchy. For every non-trivial community at every level, an asynchronous map-reduce prompt asks an LLM to emit a two-sentence summary. The persistent artifacts are (i) the FAISS index, (ii) hierarchy tensors `partitions[`L`]`, and (iii) the summary file; no similarity graph is ever created.

Retrieval. Our inference-time pipeline mirrors the two-tier strategy popularised by Microsoft's GraphRAG, but operates entirely in vector space. **Local search** begins at the *bottom* of the VLouvain hierarchy (fine communities that each contain a few passages). Given a query q we embed it once to q_e *(Alg. 1, line 2)*, compute the cosine similarity between q_e and every community summary on that bottom layer, and keep the most related communities *(line 3)*. Only the member passages of those commu-

Algorithm 1 Retrieval logic used by GraphRAG-V (simplified pseudocode)

```
1: function LOCAL(q)
2:     q_e ← EMBED(q)                                   ▷ single query embedding
3:     identify the k_c most similar fine communities to q_e
4:     gather every passage from those communities
5:     rank the passages by cos(q_e, v_i) and keep the first n_p
6:     return LLM(q | selected passages)
7: end function
8: function GLOBAL(q)
9:     q_e ← EMBED(q)
10:    pick the top three root communities by q_e^⊤ summary-vector
11:    for each chosen root do
12:        compose a prompt from its child summaries
13:        partial ← LLM(q | child summaries)
14:    end for
15:    return LLM_fuse(all partial answers)
16: end function
```

nities are inspected further: we rank their embeddings against q_e *(line 4)*, select passages that fit a fixed token budget *(line 5)*, and send a single prompt *(question + selected passages)* to the LLM *(line 6)*. Focusing the search space in this way retains the speed of classic vector RAG while steering the model toward tightly related evidence.

Global search starts at the *top* of the hierarchy. We embed the same query vector q_e *(line 8)* against the summaries of the root-level communities (each root summarises hundreds of passages spread across the corpus). The three most relevant roots are chosen *(line 9)*. For each root we gather the summaries of its immediate children *(line 10)*, compose them into a short "cluster context", and ask the LLM for a *partial* answer that relies only on those summaries *(line 11)*. Finally, a single FUSE prompt shows all partial answers to the LLM and asks it to merge them into one coherent response, resolving overlap or conflicts *(line 12)*.

Running *both* paths for every query yields complementary strengths: the local branch excels when the answer is concentrated in a few passages, whereas the global branch shines when evidence is scattered across distant parts of the corpus (e.g. comparison or temporal questions). The additional cost of the global branch is negligible compared to LLM generation time. It requires only one extra embedding step, three short partial prompts, and a final fusion prompt, yet consistently improves answer quality in our experiments.

3.3 Complexity Analysis

For n chunks and embedding dimension d the pipeline's cost is

Component	Time	Extra memory
Embedding	$O(nd)$	$O(nd)$
VLouvain (L sweeps)	$O(ndL)$ GPU	$O(nd)$
Community summaries	$O(\|\mathcal{C}\|)$ LLM calls	$O(\|\mathcal{C}\|)$
ANN lookup (per query)	$O(\log n)$	—

Advantages. GraphRAG-V removes entity extraction, JSON-triple storage, k-nearest pruning, and the quadratic edge list, yielding four concrete gains. (i) *Time efficiency*: indexing completes in minutes and query-time retrieval is dominated by a single FAISS lookup plus one-two LLM calls, whereas graph pipelines spend seconds parsing a query and traversing a materialised graph. (ii) *Token and cost efficiency*: the indexing stage issues zero LLM prompts, eliminating the hundreds (or thousands) of calls required to extract entities and relations in GraphRAG-style systems. (iii) *Memory efficiency*: the working set stays at $O(nd)$, consisting of two $n \times d$ matrices and two length-n vectors, compared to the potentially $O(n^2)$ edge list required by explicit graphs. (iv) *No information loss*: all pairwise similarities remain available, so long-range evidence is never discarded by pruning heuristics. Together these properties allow GraphRAG-V to deliver broader context, higher answer accuracy, and markedly faster end-to-end speed, all while imposing only a fraction of the engineering overhead borne by graph-based RAG competitors.

4 Experimental Setup

The evaluation is conducted on the MultiHopRAG benchmark [26], which supplies 2556 questions spanning four categories: *comparison* (33.5 %), *inference* (31.9 %), *temporal* (22.8 %), and *null* (11.8 %). We adopt this benchmark precisely because each question demands reasoning over evidence scattered across multiple documents, especially the *comparison* queries, which require synthesizing facts drawn from several distinct sources. This makes it an ideal test bed for a system designed to aggregate information from many resources.

Two backbone models are used: **Qwen 2.5-7B** [27] and **Llama 3.1-8B** [9]. On the full question set we measure: (i) Zero-shot generation and (ii) VectorStore-RAG (600-token chunks with 100-token overlap, BGE-m3 embeddings) for both LLMs, and (iii) our *GraphRAG-V* pipeline for Qwen. Because GraphRAG's open-source implementation exceeded our single-GPU resources (Section 5), we compare against Microsoft's local and global graph pipelines on a stratified **125-question subset**. The subset preserves the original class ratios.

Answer quality is assessed with *Accuracy* and *Recall* from the benchmark scorer. Indexing time and total question-answering time quantify efficiency. GraphRAG-V always returns both local and global communities in a single prompt, so quality is reported once per question.

All runs use a single Google Colab instance (A100 40 GB, 83.5 GB RAM, 235 GB SSD).

Table 1: Indexing/setup time on the MultiHopRAG corpus.

Method	Wall-clock time
VectorStore index (shared)	74 s
+ VLouvain community discovery	5.1 s
+ community summarisation	4 min
GraphRAG-V total	5.3 min
Microsoft GraphRAG: initiation	40 s
+ full graph construction	≈3 h
Microsoft GraphRAG total	3.0 h

Table 2: Wall-clock time to answer all 2556 questions. ◊ = extrapolated from partial run.

System	Total run time
Zero-shot Qwen 2.5	6 min
VectorStore-RAG (Qwen 2.5)	12 min
GraphRAG-V (local+global, Qwen 2.5)	42 min
Microsoft GraphRAG local ◊	≈28 h
Microsoft GraphRAG local+global ◊	≈144 h

5 Results

We first compare *efficiency*: how long each pipeline takes to prepare an index and to answer the entire 2 556-question suite. All numbers are measured on the single-GPU Colab node described in Section 4. Microsoft GraphRAG figures marked ◊ are extrapolated from the observed processing rate before the run terminated owing to disk exhaustion.

Table 1 shows that GraphRAG-V completes indexing in 5.3 min, almost two orders of magnitude faster than the 3 h required by Microsoft's graph pipeline. The additional overhead beyond the shared vector index is dominated by a one-off summary pass.

As summarised in Table 2, GraphRAG-V answers the entire question set in 42 min; the added cost over VectorStore-RAG stems from running both local and global retrieval for every query. In contrast, the GraphRAG implementation would require an estimated 28 h for local retrieval alone and 144 h when local and global modes are combined, well beyond our compute budget.

Because GraphRAG-V is lightweight, *both* local and global contexts are retrieved for every query and supplied jointly to the LLM.

GraphRAG-V boosts overall recall from 37.9 % (vector baseline) to **48.8 %** while adding four points of accuracy.

On the 125-query subset (small enough to fit our single-GPU time and compute budget), GraphRAG-V outperforms Microsoft's graph pipeline in both accuracy and recall: it scores +5.1 pp / +12.0 pp above GraphRAG-Local and +6.9 pp / +16.8 pp above GraphRAG-Global. The speed gap

Table 3: Accuracy / Recall (%) for the full 2556-question corpus.

	Inference		Comparison		Null		Temporal		Overall	
Method	A	R	A	R	A	R	A	R	A	R
Llama-Zero shot	42.9	33.3	33.5	0.8	33.3	0.0	33.7	1.7	36.1	11.3
Llama-Vector RAG	64.8	72.8	34.4	4.6	33.7	1.7	34.6	5.7	40.4	26.3
Qwen-Zero shot	58.5	64.5	34.0	3.0	34.3	4.3	35.9	10.6	39.9	24.5
Qwen-Vector RAG	81.4	88.6	35.5	9.0	39.4	23.3	37.6	17.0	44.6	37.9
GraphRAG-V	**86.3**	**92.0**	**38.8**	**21.0**	**39.9**	**24.6**	**46.1**	**41.5**	**49.4**	**48.8**

Table 4: Overall accuracy / recall (%) on the 125-question subset.

Method	Accuracy	Recall
Zero-shot (Qwen)	39.9	24.5
Vector RAG (Qwen)	47.9	45.6
GraphRAG-Local (Qwen)	43.9	36.0
GraphRAG-Global (Qwen)	42.1	31.2
GraphRAG-V (Qwen)	**49.0**	**48.0**

is even larger: our method finishes the batch in about 2 minutes, whereas the local-only variant takes roughly 55 minutes and the combined global pass extends to about 6 hours 25 minutes. Thus GraphRAG-V delivers higher answer quality while cutting end-to-end latency by more than an order of magnitude.

6 Discussion

Our results show consistent gains over both zero-shot baselines and classic vector-store RAG, but the *magnitude* of the improvement varies by question type. The largest jump appears in the *comparison* and *temporal* categories, where answers must synthesise observations that lie in different parts of the corpus or order events along a timeline. GraphRAG-V excels in these settings because the global branch starts its search at the very top of the hierarchy, ensuring that evidence drawn from distant regions of the corpus is considered *before* passage-level ranking shrinks the context window. By contrast, classic vector retrieval ranks passages independently; near-duplicate snippets crowd out complementary facts, and long-range relationships are missed. Our local branch recovers the needle-in-a-haystack strength of standard RAG, so the joint pipeline maintains parity on single-hop *inference* questions while delivering a pronounced lift when broader context is required.

Why explicit graphs struggle. Graph-based RAG systems promise global context through entity-relation graphs, yet they introduce two practical complications. First, edge weights are themselves generated by an LLM. Although described as "deterministic", the scores vary with the

model, the prompt template, and even random sampling noise, leading to edge-weight ranges that differ by an order of magnitude across corpora. Such variability can degrade community detection quality. For example, it perturbs modularity in Louvain-style objectives and generates a large volume of intermediate JSON artifacts, which increases disk and memory usage. Second, state-of-the-art implementations prune the fully connected graph with k-nearest-neighbour heuristics to keep memory manageable. Pruning removes as many as $n^2 - kn$ edges, severing the very long-distance links that multi-hop reasoning depends on; the damage is worst for comparison questions that hinge on relationships between otherwise distant entities.

Community structure without compromise. GraphRAG-V sidesteps both issues. Similarities are implicit dot products between unit embeddings—numerically stable, model-agnostic, and reproducible. VLouvain maintains *all* pairwise affinities without materialising an edge list, so no k-NN pruning is needed and no information is lost. Empirically this design closes the accuracy gap on inference queries while more than doubling precision and recall on comparison and temporal questions.

Efficiency in practice. Because indexing requires only one GPU pass over the embedding matrix, our end-to-end setup time is minutes rather than hours, and query latency adds one partial-answer map-reduce on top of a standard RAG call—negligible next to LLM generation time. Memory footprints stay at $O(nd)$, making the method tractable on a single A100 even for million-passage corpora. Taken together, these properties show that *chunk-similarity community structure, not the construction of knowledge graphs, is key to broad context reasoning*, and that it can be leveraged with significantly less cost and engineering effort than previously thought.

7 Conclusion & Future Work

Existing retrieval-augmented generation pipelines sit at two extremes: vector-store RAG is lightning-fast but falters when answers require evidence dispersed across many documents, whereas graph-based RAG captures global context yet pays a heavy price in LLM calls, graph construction time, and memory. GraphRAG-V bridges this gap. By applying the edge-free *VLouvain* algorithm directly to chunk embeddings we uncover multi-level *vector communities* without materialising a graph, preserving every pairwise relationship while adding only $O(nd)$ memory. Combined local + global retrieval delivers markedly higher accuracy, especially on comparison and temporal questions, at a fraction of the indexing and query latency of GraphRAG. The entire system runs on a single GPU, requires no costly entity extraction prompts, and achieves state-of-the-art performance with modest LLMs.

Future work will test this approach at even larger scales (tens of millions of chunks) and on resource-constrained hardware such as laptop GPUs or edge accelerators, further validating its efficiency advantages.

References

1. Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al.: Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023)
2. Akesson, S., Santos, F.A.: Clustered retrieved augmented generation (crag). arXiv preprint arXiv:2406.00029 (2024)
3. Anonymous: Vlouvain: Graph-free community detection for large vector collections. In: Proceedings under Double-Blind Review (2025), preprint to be released post-review
4. Asai, A., Wu, Z., Wang, Y., Sil, A., Hajishirzi, H.: Self-rag: Learning to retrieve, generate, and critique through self-reflection. In: 12th International Conference on Learning Representations (2023)
5. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of statistical mechanics: theory and experiment **2008**(10), P10008 (2008)
6. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. NeurIPS **33**, 1877–1901 (2020)
7. Chen, J., Xiao, S., Zhang, P., Luo, K., Lian, D., Liu, Z.: Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation (2023)
8. Edge, D., Trinh, H., Cheng, N., Bradley, J., Chao, A., Mody, A., Truitt, S., Metropolitansky, D., Ness, R.O., Larson, J.: From local to global: A graph rag approach to query-focused summarization. arXiv preprint arXiv:2404.16130 (2024)
9. Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., et al.: The llama 3 herd of models. arXiv preprint arXiv:2407.21783 (2024)
10. Guo, Z., Xia, L., Yu, Y., Ao, T., Huang, C.: Lightrag: Simple and fast retrieval-augmented generation (2024)
11. Gutiérrez, B.J., Shu, Y., Gu, Y., Yasunaga, M., Su, Y.: Hipporag: Neurobiologically inspired long-term memory for large language models. In: NeurIPS (2024)
12. Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., et al.: A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. ACM Transactions on Inf. Systems **43**(2), 1–55 (2025)
13. Huang, Y., Zhang, S., Xiao, X.: Ket-rag: A cost-efficient multi-granular indexing framework for graph-rag. arXiv preprint arXiv:2502.09304 (2025)
14. Karpukhin, V., Oguz, B., Min, S., Lewis, P.S., Wu, L., Edunov, S., Chen, D., Yih, W.t.: Dense passage retrieval for open-domain question answering. In: EMNLP (1). pp. 6769–6781 (2020)
15. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.t., Rocktäschel, T., et al.: Retrieval-augmented generation for knowledge-intensive nlp tasks. NeurIPS **33**, 9459–9474 (2020)
16. Liu, N.F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., Liang, P.: Lost in the middle: How language models use

long contexts. Transactions of the Association for Computational Linguistics **12**, 157–173 (2024)
17. Ma, S., Xu, C., Jiang, X., Li, M., Qu, H., Yang, C., Mao, J., Guo, J.: Think-on-graph 2.0: Deep and faithful large language model reasoning with knowledge-guided retrieval augmented generation. arXiv preprint arXiv:2407.10805 (2024)
18. Malkov, Y.A., Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE transactions on pattern analysis and machine intelligence **42**(4), 824–836 (2018)
19. Mavromatis, C., Karypis, G.: Gnn-rag: Graph neural retrieval for large language model reasoning. Preprint arXiv:2405.20139 (2024)
20. Newman, M.E.: Modularity and community structure in networks. PNAS **103**(23), 8577–8582 (2006)
21. Ram, O., Levine, Y., Dalmedigos, I., Muhlgay, D., Shashua, A., Leyton-Brown, K., Shoham, Y.: In-context retrieval-augmented language models. Transactions of the Association for Computational Linguistics **11**, 1316–1331 (2023)
22. Reimers, N., Gurevych, I.: Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084 (2019)
23. Santhanam, K., Khattab, O., Saad-Falcon, J., Potts, C., Zaharia, M.: Colbertv2: Effective and efficient retrieval via lightweight late interaction. arXiv preprint arXiv:2112.01488 (2021)
24. Shi, W., Min, S., Yasunaga, M., Seo, M., James, R., Lewis, M., Zettlemoyer, L., Yih, W.t.: Replug: Retrieval-augmented black-box language models. arXiv preprint arXiv:2301.12652 (2023)
25. Song, K., Tan, X., Qin, T., Lu, J., Liu, T.Y.: Mpnet: Masked and permuted pre-training for language understanding. NeurIPS **33**, 16857–16867 (2020)
26. Tang, Y., Yang, Y.: Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries. arXiv preprint arXiv:2401.15391 (2024)
27. Team, Q.: Qwen2.5: A party of foundation models (September 2024), https://qwenlm.github.io/blog/qwen2.5/
28. Traag, V.A., Waltman, L., Van Eck, N.J.: From louvain to leiden: guaranteeing well-connected communities. Scientific reports **9**(1), 1–12 (2019)
29. Wang, S., Fang, Y., Zhou, Y., Liu, X., Ma, Y.: Archrag: Attributed community-based hierarchical retrieval-augmented generation. arXiv preprint arXiv:2502.09891 (2025)
30. Xiao, S., Liu, Z., Zhang, P., Muennighoff, N.: C-pack: Packaged resources to advance general chinese embedding (2023)
31. Xiao, S., Liu, Z., Zhang, P., Xing, X.: Lm-cocktail: Resilient tuning of language models via model merging (2023)
32. Yang, A., et al.: Qwen2 technical report. arXiv preprint arXiv:2407.10671 (2024)
33. Zhang, P., Xiao, S., Liu, Z., Dou, Z., Nie, J.Y.: Retrieve anything to augment large language models (2023)
34. Zhao, P., Zhang, H., Yu, Q., Wang, Z., Geng, Y., Fu, F., Yang, L., Zhang, W., Jiang, J., Cui, B.: Retrieval-augmented generation for ai-generated content: A survey. preprint arXiv:2402.19473 (2024)