

Don't Forget the Base Retriever! A Low-Resource Graph-based Retriever for Multi-hop Question Answering

André Melo[§] Enting Chen^{¶†} Pavlos Vougiouklis[§] Chenxin Diao[§]

Shriram Piramanayagam[§] Ruofei Lai[§] Jeff Z. Pan[§]

[§]Huawei UK R&D Technologies, Poisson Lab, CSI [¶]Baillie Gifford

{andre.melo, pavlos.vougiouklis, chenxindiao}@huawei.com

enting.chen@bailliegifford.com

{shriram.piramanayagam, lairuofei, jeff.pan}@huawei.com

Abstract

Traditional Retrieval-augmented Generation systems struggle with complex multi-hop questions, which often require reasoning over multiple passages. While GraphRAG approaches address these challenges, most of them rely on expensive LLM calls. In this paper, we propose GRIEVER, a lightweight, low-resource, multi-step graph-based retriever for multi-hop QA. Unlike prior work, GRIEVER does not rely on LLMs and can perform multi-step retrieval in a few hundred milliseconds. It efficiently indexes passages alongside an associated knowledge graph and employs a hybrid retriever combined with aggressive filtering to reduce retrieval latency. Experiments on multi-hop QA datasets demonstrate that GRIEVER outperforms conventional retrievers and shows strong potential as a base retriever within multi-step agentic frameworks.

1 Introduction

Recent efforts have highlighted the benefits of combining Large Language Models' (LLMs) parametric memory with non-parametric sources of knowledge to address complex user queries. The Retrieval-augmented Generation (RAG) paradigm exemplifies this approach, using non-parametric memory to guide LLMs towards more accurate responses (Lewis et al., 2020). While effective for simpler queries, multi-hop question answering (QA) remains a more demanding task, requiring reasoning across multiple passages or documents.

Recent GraphRAG approaches have proposed more sophisticated retrieval strategies that bridge the semantic meaning of passages (or *documents*) by interlinking the entities they contain (Fang et al., 2024; Li et al., 2024; Gutierrez et al., 2024; Shen et al., 2025). They usually work by leveraging

an alignment of an index of passages with an index of triples extracted from these passages (Fang et al., 2024; Gutierrez et al., 2024; Li et al., 2024; Shen et al., 2025). However, such methods often rely on expensive LLM calls, incurring significant deployment costs, limiting their applicability in resource-constrained environments.

In this work, we focus on efficient retrieval and we propose: GRIEVER, a graph-based retriever that resolves multi-hop challenges without relying on online LLM calls. We take the alignment between a passage and triples index to the extreme, and we propose a hybrid indexing paradigm on top of the structure of the extracted graph tailored to the needs of *partial* triples retrieval and synonym expansion. We build a lightweight, graph-based retrieval strategy using this hybrid index that expands triple reasoning chains to approximate suitable relevant passages across distant semantic hops. Low-latency and results filtering considerations are applied to reduce the computational retrieval load.

We evaluate GRIEVER on popular multi-hop QA datasets: MuSiQue, 2Wiki and HotpotQA. GRIEVER achieves superior passage recall performance compared to other conventional base retrievers, and in many cases comparable to LLM-based GraphRAG approaches, while running at least an order of magnitude faster. Furthermore, we find that in settings with more flexible runtime requirements, when GRIEVER can be combined with agentic frameworks (i.e. GEAR) to provide relative recall gains of up to 4.9%. Our contributions can be summarised as follows:

- We refine the GraphRAG paradigm of aligning a passage and triples index by introducing efficiency and robustness considerations for hybrid passage and *partial* triple retrieval.
- We propose an iterative methodology for expanding triples associated with a preliminary

[†]Work done while at Huawei Edinburgh Research Centre.

set of retrieved passages by considering filtering unsuitable candidate entity nodes.

- We demonstrate the value of GRIEVER within multi-step, agentic frameworks enabling performance convergence within less iterations.

2 Related Work

Our work draws inspiration from multi-hop QA using combinations of LLMs with graphical structures. In recent years, several architectures introduced a separate, offline indexing phase during which they form a hierarchical representation of the information included in a knowledge of textual passages (Chen et al., 2023; Sarthi et al., 2024; Edge et al., 2024).

Fang et al. construct reasoning chains through an auto-regressive reasoning chain constructor, and generate answers either directly from these reasoning chains or by retrieving original context documents. Li et al. use an LLM agent that is capable of selecting between a set of predefined actions on how to traverse the nodes of the extracted knowledge graph given an input question, in real-time. HippoRAG leverages an alignment of passages and extracted triples in order to retrieve passages based on the Personalised PageRank algorithm (Gutierrez et al., 2024). While the proposed retriever results in considerable improvements for single- and multi-step retrieval (i.e. when coupled with IR-CoT (Trivedi et al., 2023)), it remains agnostic to the semantic relationships of the extracted triples. Shen et al. introduce a new graph-based retrieval framework that uses a small semantic model for exploring multi-hop relationships. While the proposed system: GEAR leads to a reduction in LLM token utilisation, similarly to HippoRAG, it still relies on an LLM for its retrieval step—leading to significant runtimes.

In this paper, we build upon a similar alignment of passages and extracted triples, refining the approach by incorporating retrieval efficiency considerations and enabling the retrieval of partial triples, without relying on LLMs for addressing multi-hop QA scenarios. Our revision of conventional GraphRAG approaches enables retrieval runtimes of under one second while maintaining competitive performance compared to significantly more computationally expensive alternatives.

3 Problem Setting

The problem involves multi-step passage retrieval, where a given question requires the information from multiple passages to be combined in order to answer a question (Trivedi et al., 2022; Yang et al., 2018; Ho et al., 2020). Additionally, we assume an alignment of an index containing these passages with an index of triples extracted from these passages (Trivedi et al., 2023; Gutierrez et al., 2024; Shen et al., 2025). These triples represent atomic facts within their source passages. They are subsequently organised into a graph bridging passages sharing common entities.

We approach the problem from a low-resource perspective, where multi-hop capabilities are required but results must be returned within hundreds of milliseconds (100 – 500 ms).

4 GRIEVER

Similar to other works in the RAG space, GRIEVER relies on the existence of a corpus of passages and associated graph triples, which can be extracted directly from the passages. GRIEVER introduces an offline index building step (described in detail in Section 4.1). The relevant considerations go beyond conventional alignments of passages and triples indices, ensuring efficient retrieval capabilities during online, querying time, which is subsequently described in Section 4.2.

4.1 Offline Index Construction

The overall architecture of GRIEVER includes an initial offline step, where the index structure is created. GRIEVER uses three different indices:

- **passages:** contains the uniquely identified passages with their document title, passage content and associated triples information;
- **partial_triples:** contains partial triples, i.e., subject-predicate or predicate-object pairs alongside the passage ids from which the triple partials were extracted and the list of partial’s complement entities; and,
- **same_as:** contains a list of synonyms along with a list of passage ids from which these synonyms are extracted.

Further information about the structure of the indices used by GRIEVER is provided in Appendix A.1. The role each index plays within the

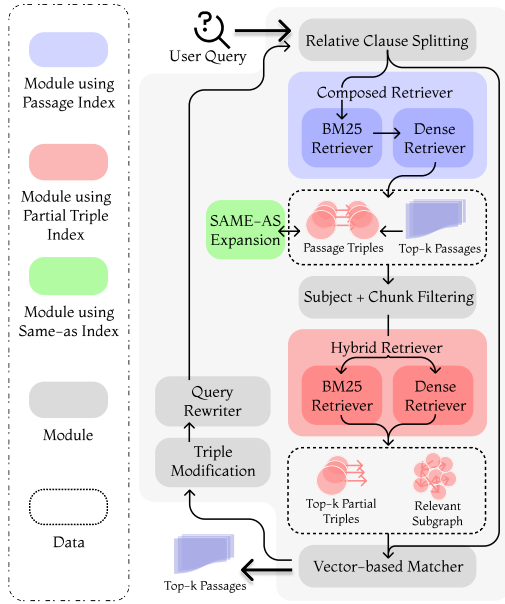


Figure 1: The architecture of GRIEVER.

retrieval framework is shown in Figure 1 and explained in details in Section 4.2

4.2 Online Multi-Step Retrieval

Once the offline index construction is concluded, the indices can be used by an online multi-step iterative retrieval system, c.f. Figure 1. Algorithm 1 shows the iterative retrieval process in further detail. Figure 2 shows an example of a multi-hop query requiring information from two different passages. Each online iteration of GRIEVER achieves its functionality following three retrieval steps.

- The first retrieves the passages alongside their associated triples given a query by relying on the passages index.
- The second step retrieves synonyms of entities using the same_as index to improve path recall by enabling the joining of entities that appear in triples connected to the sub-graph retrieved in the preceding step.
- The third one shortlists the triples obtained with passages using the partial_triples by selecting those scored the highest with respect to the original text query.

4.2.1 Relative Clause Splitting

The first part of the pipeline is a lightweight relative clause splitter. Relative clause connectors (shown in bold in the example query below) are used to split the query and the split parts, e.g.:

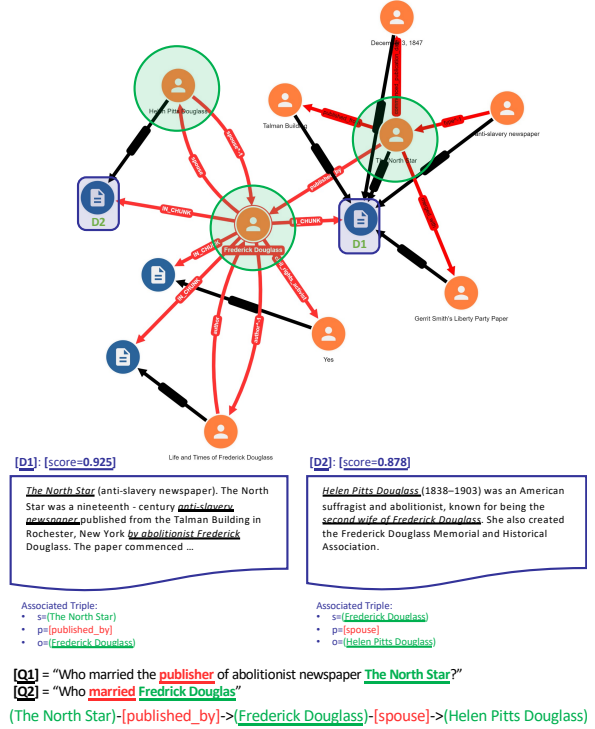


Figure 2: Example of an input query for GRIEVER.

q = “When did Napoleon occupy the city **where** the mother of the woman **who** brought Louis XV style to the court died?”

Then for each split point, a new sub-query is generated by joining the parts from the split point until the end. Those sub-queries are fired in parallel and their results are combined at the end.

The benefit of splitting the query is that for queries with multiple relative clauses, each clause may correspond to a different passage to be retrieved. Using the entire query in the first iteration can introduce mixed signals, making it difficult to retrieve the first hop. This is a lightweight yet effective way of addressing this issue.

4.2.2 Passage Retrieval

Given a query, the first step is to retrieve the set of best-matching passages. Hybrid retrievers are able to combine the benefits of both sparse and dense retrievers. Reciprocal Rank Fusion (RRF) is the most widely used method to combine results from both dense and sparse retrievers (Cormack et al., 2009). While the underlying retrievers are independent and can be called in parallel, the dense retriever is normally much slower than the sparse one. The latency of the dense retriever primarily depends on the dimensionality of the embedding vectors and the number of candidates for which similarity is computed (Malkov and Yashunin, 2020).

In order to minimise the latency of the dense retriever, we propose a Composed retrieval strategy which is presented in detail below.

Optimising Hybrid Retrieval Latency In order to further optimise the runtime of each base retrieval call within GRIEVER, we minimise the number of candidates that will be considered by the dense retrieval by relying on a candidate set filter based on the results of the sparse retriever. As we show in our experiments (see Table 1), the overall runtime is often lower, as the benefit of parallelising the sparse retriever does not compensate for the speed-up due to filtering.

When aggregating results from the dense and sparser retrievers, RRF is a robust popular option, which ignores scores and relies on rankings only. Since the passage retrieval scores can be used in the final ranking and combined with other types of scoring, it makes sense to use more meaningful and comparable scores. The details about the passage scoring are presented in the next section.

In the first iteration, the only input is the query itself. After the first iteration, there are filtering opportunities which can be used to further reduce the passages query runtime. The first iteration produces a sub-graph, which can be extended given a rewritten query at a subsequent iteration (c.f. Section 4.5). For the sub-graph extension, it is assumed that any *new passages* should contain triples that connect them with the existing sub-graph. That is, the subjects or objects of the triples should match those in the edges of the sub-graph. The set of *joinable entities* in the sub-graph can be used to filter the passages, ensuring that at least one of them is contained in the subjects or objects field in the passage index (see Appendix A.1). Additionally, the `passage_id` set can be used as an inverse filter to ensure passage novelty.

4.2.3 Join Entities Synonym Expansion

In the context where the graph is formed by text triplets, the aforementioned subjects or objects filter would miss cases where the join entity appears on different triples as different aliases, e.g., “LeBron James” and “LeBron Raymone James Sr.”.

In order to address such cases and improve coverage, we consider the `same_as` index in order to expand the sub-graph of the joined entities with their synonyms. The set of already retrieved `passage_ids` can be used to filter the `same_as` query, so that only synonyms relevant to the in-

tended meaning of the entity (i.e. based on the context in those passages) are retrieved. This helps avoid retrieving synonyms related to an alternative or incorrect sense of the entity name. For instance, without `passage_id` filtering, the name Orange may be expanded with synonyms corresponding to different entities, such as a company, a fruit, or something else entirely.

4.2.4 Triples Shortlisting

Once the top- k passages are retrieved, their associated triples can be directly obtained without an extra query, since they are included in triples field of the `partial_triples` index. Since each passage may contain several triples, the total number of triples retrieved can be quite high. This may be problematic, as the final triple scoring relies on an efficient vector-based matcher that requires the entities in each triple to be encoded (Stoilos et al., 2022). Consequently, the encoding operation can become costly if too many triples are considered.

The `partial_triples` is used to shortlist the set of triples considered in the unsupervised tagger, c.f. Algorithm 2 for further details. This index is used instead of a full `triples_index`, since in the QA setting the whole triple is not expected to appear within an input natural language query (Mohammed et al., 2018). Instead, only a combination of a predicate (or property) and a subject or object is expected to be contained in the original query. The example in Figure 2 illustrates this. The first hop information of “Who married the publisher of abolitionist newspaper The North Star?” is supposed to be addressed by the triple (The North Star)-[published_by]->(Frederick Douglass). The entity Frederick Douglas is the example of the required joining entity, which is originally unknown and will be included to join the first hop triples with the ones for the next hop (Frederick Douglass)-[spouse]->(Helen Pitts Douglass).

In the `partial_triples` index, the concatenation of subject + predicate and predicate + object (if an inverse partial triple) is indexed for computing the relevant dense retriever embeddings.

4.3 Vector-based Entity Matcher

We use a lightweight vector-based matcher on the token-level embeddings, enc_t^q , of a sentence encoder. The embeddings for an n -gram, n_g , derived from q , are computed after aggregating the corresponding n_g token-level embeddings as follows:

	Retriever	R@5	R@10	R@15	ms@5	ms@10	ms@15
MuSiQue	BM25	0.351	0.41	0.442	23±5	25±6	27±6
	Dense	0.319	0.383	0.420	294±12	292±11	292±11
	RRFHybrid	0.394	0.472	0.505	295±11	295±10	294±10
	Composed	0.417	0.492	0.533	135±8	134±7	168±13
	GRIEVER	0.456	0.539	0.573	425±122	441±135	509±127
2Wiki	BM25	0.64	0.668	0.68	38±15	44±16	48±17
	Dense	0.467	0.515	0.539	467±24	467±24	472±23
	RRFHybrid	0.64	0.673	0.685	469±24	472±25	475±24
	Composed	0.624	0.662	0.677	159±22	229±26	297±29
	GRIEVER	0.676	0.738	0.751	435±93	476±77	603±77
HotpotQA	BM25	0.668	0.82	0.887	18±3	21±3	24±3
	Dense	0.728	0.799	0.842	61±3	61±3	62±3
	RRFHybrid	0.776	0.879	0.917	63±3	61±3	63±3
	Composed	0.784	0.886	0.919	76±5	96±8	115±7
	GRIEVER	0.813	0.909	0.937	273±59	300±49	337±53

Table 1: Retrieval performance and runtime comparison on MuSiQue, 2Wiki and HotpotQA.

$\text{enc}(n_g|\mathbf{q})$. This means that the rather expensive encoding step happens once for each $\mathbf{q}$. $\text{enc}(\mathbf{q})$ is used for dense retrieval and the intermediary $\text{enc}_t^{\mathbf{q}}$ are reused to compute the n -gram embeddings.

The resulting n -gram embeddings are used to compute similarity scores against the embeddings of the named entities. The candidate named entities come from the subjects or objects of shortlisted triples. If synonyms exist for a given entity e , each synonym is encoded separately. We denote the *entity matching score*, $s_e^{\mathbf{q}}$, as the maximum similarity score observed for an entity and its synonyms across all candidate n -grams derived from $\mathbf{q}$.

4.4 Scoring: From Triples to Paths and Passages

Let a triple t_j be denoted as $t_j = (s_j, p_j, o_j)$, its corresponding partial triples as $t'_j = \{(s_j, p_j), (p_j, o_j)\}$ and $\{s_j, o_j\}$ be the set containing the two entities in t_j . The score of a partial triple $u \in t'_j$ can be computed as the average of its predicate score, $s_{p_j}^{\mathbf{q}}$ and its entity score $s_{e_j}^{\mathbf{q}}$ s.t. $e_j \in \{s_j, o_j\}$ multiplied by the inverse partial triple frequency:

$$s_u^{\mathbf{q}} = \frac{s_{e_j}^{\mathbf{q}} + s_{p_j}^{\mathbf{q}}}{2(1 + \log \text{freq}(e_j, p_j))}, \quad (1)$$

where $e_j \in u \cap \{s_j, o_j\}$. The triple scores are computed as the maximum score across their corresponding partial triples, as follows:

$$s_{t_j}^{\mathbf{q}} = \max(\{s_u^{\mathbf{q}} \forall u \in t'_j\}) \quad (2)$$

The multi-hop path scores are updated every time a new triple is added to the path chain. The

running score of a path is defined as the average of the scores of the triples composing the path. For determining the score of a passage, we iterate over all the paths containing a triple associated with it. The score of a passage is the maximum score of the multiplication of the triple score itself with the score of each path in which this triple participates. Those passage scores are used to rank the final GRIEVER results.

4.5 Heuristic-based Query Re-writing

After identifying the highest-scoring entities using the vector-based matcher (see Section 4.3), we attempt to rewrite the query from the previous iteration by removing mentions of matched entities, along with relations from the shortlisted triples when they are matched. If successful, we replace the identified mention in the query with the remaining unmatched entity in the corresponding triples.

Retriever	MuSiQue		2Wiki		HotpotQA	
	EM	F1	EM	F1	EM	F1
BM25	18.4	27.6	42.4	47.7	43.8	57.2
Dense	15.2	25.4	23.6	28.4	42.1	55.0
RRFHybrid	20.0	30.2	43.8	48.0	45.4	58.5
Composed	19.0	29.5	43.8	47.7	43.3	57.1
GRIEVER	21.6	32.6	42.0	48.2	46.2	59.8

Table 2: End-to-end QA performance with top- 5 retrieved passages.

5 Experiments

We conduct experiments on three popular public multi-hop question answering benchmarks: MuSiQue (Trivedi et al., 2022), HotpotQA (Yang

Setup	MuSiQue			2Wiki			HotpotQA		
	R@5	R@10	R@15	R@5	R@10	R@15	R@5	R@10	R@15
w/ partial_triples	0.456	0.539	0.573	0.676	0.738	0.751	0.813	0.909	0.937
w/ full_triples	0.457	0.536	0.570	0.667	0.722	0.740	0.787	0.900	0.932
w/o shortlisting	0.452	0.532	0.574	0.594	0.695	0.713	0.739	0.870	0.916
w/o composed_retriever	0.438	0.514	0.555	0.672	0.733	0.731	0.792	0.887	0.920

Table 3: Ablation study of GRIEVER’s performance across different index configurations; with a partial triples index, with a full instead of a partial triples index, without triples shortlisting and with RRFHybrid instead of Composed retriever for passage retrieval.

Retriever	MuSiQue			2Wiki			HotpotQA		
	R@5	R@10	R@15	R@5	R@10	R@15	R@5	R@10	R@15
GEAR + RRFHybrid	55.6	66.1	69.3	88.4	94.5	95.2	93.1	96.3	96.7
GEAR + GRIEVER	55.7	66.6	69.8	88.3	94.7	95.7	92.5	95.7	96.9
IRCoT + RRFHybrid	52.57	58.30	59.98	76.00	80.60	—	79.55	82.55	—
IRCoT + GRIEVER	54.98	61.03	62.58	77.00	83.80	—	80.10	86.60	—

Table 4: Retrieval performance of GRIEVER against RRFHybrid as base retriever within agentic, multi-step frameworks (i.e. GEAR and IRCoT).

et al., 2018), and 2WikiMultiHopQA (2Wiki) (Ho et al., 2020). For MuSiQue and 2Wiki, we use the data provided in the IRCoT paper (Trivedi et al., 2023) which includes the full corpus, while for HotpotQA, we follow the same setting as HippoRAG (Gutierrez et al., 2024) to limit experimental costs.

We measure both retrieval and QA performance. For retrieval evaluation, we use Recall@ k (R@ k) for $k \in \{5, 10, 15\}$. Following standard practices, QA performance is evaluated with Exact Match (EM) and F1 scores (Trivedi et al., 2023).

5.1 Baselines

We evaluate GRIEVER against stand-alone, single-step retrievers: (i) BM25, (ii) Dense (using SBERT), (iii) a hybrid approach combining BM25 and SBERT through RRF (i.e. RRFHybrid) and (iv) the Composed retrieval strategy described in Section 4.2.2.

5.2 Retrieval and QA

The retrieval results in Table 1 indicate that GRIEVER with two iterations (i.e. with a single query re-writing step) is able to consistently outperform base retrievers across all settings and datasets. This is achieved, without a major difference in average runtime, especially when compared with the base Dense retriever. In fact, in one of the settings (top-5 on 2Wiki), GRIEVER is on average faster than Dense. It is important to note that the GRIEVER results include two dense

retrieval queries to both the passages and the partial_triples indices.

Moreover, GRIEVER’s retrieval performance is not far from that of some agentic approaches for graph-based retrieval. When comparing the retrieval results in Table 1 against the results reported by (Shen et al., 2025), we observe that GRIEVER performs comparably to, and in some settings even surpasses, IRCoT+ColBERT and HippoRAG on the MuSiQue, and HotpotQA datasets.

Table 2 confirms the hypothesis that better retrieval performance can translate into better performance in the subsequent QA phase.

Ablation Study Table 3 shows that using the partial_triples index within GRIEVER for triple shortlisting appears to improve retrieval performance. When attempting to match the entire triple against the query (by using full_triples), the absent joining entity can negatively impact the similarity score. When removing the triples shortlisting entirely (wo/ shortlisting) results in about 20–30% longer runtimes and using triples index instead of partial triples around 10–20% longer runtimes. Replacing the Composed retriever with RRFHybrid (wo/ composed_retriever) resulted in performance drop and led to about twice the runtime on average for the larger datasets MuSiQue and 2Wiki.

5.3 GRIEVER as a Base Retriever

Table 4 shows the impact of using GRIEVER as a base retriever within agentic retrieval frameworks¹: IRCOT and GEAR. GEAR uses RRFHybrid as a base retriever by default. Based on the default length of their maximum reasoning chains, the maximum number of iterations is set to 4 for MuSiQue and 2 for the HotpotQA and 2Wiki datasets. Swapping their base retriever for GRIEVER results in better retrieval performance overall.

6 Conclusion

In this paper, we introduced GRIEVER, a graph-based retriever that addresses multi-hop retrieval challenges without relying on online LLM calls. Our lightweight design leverages an alignment between a textual passage index and an index of schema-free triples, providing both efficiency and robustness.

GRIEVER streamlines passage retrieval by traversing high-scoring triple paths, enabling more effective multi-hop reasoning within hundreds of milliseconds. On popular multi-hop QA datasets, GRIEVER achieves strong performance compared to conventional base retrievers and, in many cases, rivals LLM-based GraphRAG approaches.

Finally, we demonstrate the effectiveness of GRIEVER in multi-step, agentic frameworks, where it reduces the number of iterations required to reach peak retrieval performance.

Limitations

Since GRIEVER is designed with a focus on low response time and resource efficiency, in order to remain comparable with popular baseline retrievers, it exhibits certain limitations when used in isolation. The simplicity of the query splitting, rewriting, and entity matching mechanisms can occasionally lead to incorrect triple chains and, consequently, incorrect answers. However, by relying on unsupervised methods, we ensure that the retriever remains lightweight and easily integrable into existing RAG or agentic retrieval systems.

With this in mind, there is an inherent trade-off between latency and performance when selecting sentence encoders for the query, passages, and entity matcher. Encoding constitutes the primary latency bottleneck, and the use of more complex

models can significantly impact runtime. We leave the exploration of performance using more powerful, auto-regressive semantic models to future work (Lee et al., 2025).

References

- Howard Chen, Ramakanth Pasunuru, Jason Weston, and Asli Celikyilmaz. 2023. [Walking down the memory maze: Beyond context limit through interactive reading](#). *Preprint*, arXiv:2310.05029.
- Gordon V. Cormack, Charles L A Clarke, and Stefan Buettcher. 2009. [Reciprocal rank fusion outperforms condorcet and individual rank learning methods](#). In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, page 758–759, New York, NY, USA. Association for Computing Machinery.
- Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. [From local to global: A graph RAG approach to query-focused summarization](#).
- Jinyuan Fang, Zaiqiao Meng, and Craig MacDonald. 2024. [TRACE the evidence: Constructing knowledge-grounded reasoning chains for retrieval-augmented generation](#). In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8472–8494, Miami, Florida, USA. Association for Computational Linguistics.
- Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. [HippoRAG: Neurobiologically inspired long-term memory for large language models](#). In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.
- Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. [Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps](#). In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online). International Committee on Computational Linguistics.
- Chankyu Lee, Rajarshi Roy, Mengyao Xu, Jonathan Raiman, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. 2025. [NV-embed: Improved techniques for training LLMs as generalist embedding models](#). In *The Thirteenth International Conference on Learning Representations*.
- 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. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the 34th*

¹GRIEVER is called with $k = 5$. For IRCOT, we set `max_iter=2` for 2Wiki and HotpotQA. Therefore we do not report R@15 on those datasets for IRCOT.

International Conference on Neural Information Processing Systems, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

Shilong Li, Yancheng He, Hangyu Guo, Xingyuan Bu, Ge Bai, Jie Liu, Jiaheng Liu, Xingwei Qu, Yangguang Li, Wanli Ouyang, Wenbo Su, and Bo Zheng. 2024. **GraphReader: Building graph-based agent to enhance long-context abilities of large language models**. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 12758–12786, Miami, Florida, USA. Association for Computational Linguistics.

Yu A. Malkov and D. A. Yashunin. 2020. **Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs**. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(4):824–836.

Salman Mohammed, Peng Shi, and Jimmy Lin. 2018. **Strong baselines for simple question answering over knowledge graphs with and without neural networks**. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 291–296, New Orleans, Louisiana. Association for Computational Linguistics.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D Manning. 2024. **RAPTOR: Recursive abstractive processing for tree-organized retrieval**. In *The Twelfth International Conference on Learning Representations*.

Zhili Shen, Chenxin Diao, Pavlos Vougiouklis, Pascual Merita, Shriram Piramanayagam, Enting Chen, Damien Graux, Andre Melo, Ruofei Lai, Zeren Jiang, Zhongyang Li, Ye Qi, Yang Ren, Dandan Tu, and Jeff Z. Pan. 2025. **GeAR: Graph-enhanced agent for retrieval-augmented generation**. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 12049–12072, Vienna, Austria. Association for Computational Linguistics.

Giorgos Stoilos, Nikos Papasrantopoulos, Pavlos Vougiouklis, and Patrik Banský. 2022. **Type linking for query understanding and semantic search**. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD '22*, page 3931–3940, New York, NY, USA. Association for Computing Machinery.

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

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. **Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions**. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*,

pages 10014–10037, Toronto, Canada. Association for Computational Linguistics.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. **HotpotQA: A dataset for diverse, explainable multi-hop question answering**. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

A Implementation Details

All indices are implemented using Elasticsearch. For the dense retriever, we use the all-mpnet-base-v2 SBERT model with approximate k -nearest neighbour search and cosine similarity for computing vector comparisons.

We employ GPT-4o mini for all methods that require an LLM, including triple extraction. For triple extraction, we reproduce the methodology proposed by [Gutierrez et al.](#); [Shen et al.](#).

A.1 GRIEVER’s Offline Indexing

GRIEVER uses three different indices: (i) passages, (ii) partial_triples, and (iii) same_as. The fields included in these indices are presented below:

- passages:
 - id: uid
 - doc_id: id
 - title: text
 - content: text
 - triples: list[tuple[keyword]]
 - subjects: list[keyword]
 - objects: list[keyword]
 - embedding: list[float]
- partial_triples:
 - id: uid
 - passage_ids: list[id]
 - content: text
 - subject: keyword
 - predicate: keyword
 - in_inverse: boolean
 - objects: list[keyword]
 - embedding: list[float]
- same_as:
 - id: uid
 - passage_ids: list[id]
 - content: text
 - aliases: list[keyword]
 - embedding: list[float]

A.2 GRIEVER Iterative Retrieval

Algorithms 1 and 2 illustrate the online retrieval process and triples shortlisting using the partial_triples index.

Algorithm 1 The Iterative Multi-step Retrieval Process of GRIEVER

```
function GRIEVER_RETRIEVAL( $q, k, n\_i$ )  
   $F \leftarrow \emptyset$   
   $G_q \leftarrow \emptyset$   
   $R \leftarrow \emptyset$   
  for  $i \in \text{range}(n\_i)$  do  
    if  $\neg F$  then  
       $R_i \leftarrow \text{composed\_retrieval}(q, k, \text{"passages\_index"}, F)$   
    else  
       $R_i \leftarrow \text{rrf\_retrieval}(q, k, \text{"passages\_index"}, F)$   
    end if  
     $R \leftarrow R \vee R_i$   
     $T \leftarrow \text{shortlist\_triples}(q, R_i)$   
     $(\text{scored\_triples}, E_j) \leftarrow \text{score\_triples}(q, T)$   
     $q \leftarrow \text{rewrite\_query}(q, \text{scored\_triples}, E_j)$   
     $F \leftarrow \text{filter}(\text{"passage\_ids"}, \{d.\text{passage\_id} \mid d \in R_i\}) \wedge \text{filter}(\text{"subject|object"}, E_j)$   
     $G_q \leftarrow \text{extend\_subgraph}(G_q + T)$   
  end for  
  return  $\text{rank\_passages}(R, G_q)$   
end function
```

Algorithm 2 Shortlisting Triples with the Partial Triples Index

```
function SHORTLIST_TRIPLES( $q, r, t\_max$ )  
   $T \leftarrow \{t \mid d \in r, t \in d.\text{triples}\}$   
   $E \leftarrow \{t.\text{subject} \mid t \in T\} \mid \{t.\text{object} \mid t \in T\}$   
   $E \leftarrow E \vee \text{expand\_same\_as}(E)$   
   $F \leftarrow \text{filter}(\text{"passage\_id"}, \{d.\text{passage\_id} \mid d \in r\}) \wedge$   
     $\text{filter}(\text{"subject"}, \{t.\text{subject} \mid t \in T\} \mid \{t.\text{object} \mid t \in T\})$   
   $P_T \leftarrow \text{rrf\_retrieval}(q, \text{"partial\_triples"}, t\_max, F)$   
   $T \leftarrow \{t \mid (t \in T) \wedge ((t.s, t.p) \in P_T \vee (t.o, t.p) \in P_T)\}$   
   $E_j \leftarrow \{o \mid p \in P_T, (o \in p.\text{objects}) \wedge ((p.s, p.p, p.o) \in T \vee (p.o, p.p, p.s))\}$   
  return  $(T, \text{expand\_same\_as}(E_j))$   
end function
```
