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

Introduction

- While GraphRAG approaches address complex QA scenarios, most of them rely on expensive LLM calls
- We propose GRiever: a lightweight, low-resource, multi-step graph-based retriever for multi-hop QA
- GRiever does not rely on LLMs and can perform multi-step retrieval within hundreds of milliseconds

Problem Setting

- Multi-step passage retrieval requires information from multiple passages to be combined to answer a question
- Alignments between passages and triples extracted from these passages
  - Triples represent atomic facts within their source passages
  - These triples are organised into a graph bridging passages sharing common entities
- LLM-less, lightweight setting where results must be returned within $100 - 500$ ms

A Walk-through

Offline Indexing

GRiever uses three different indices: (i) `passages`: text passages with associated list of triples, (ii) `partial_triples`: subject-predicate or predicate-object pairs and the list of complementing object or subjects and passage ids, and (iii) `same_as`: entity synonyms index .

Online Multi-step Retriever

1. Relative Clause Splitting

The first part of the pipeline is a lightweight relative clause splitter based on relative clause connectors.

2. Passage Retrieval

Conditional Hybrid Retrieval: each base retrieval call within GRiever, the number of candidates that will be considered by the dense retrieval is limited to the top-$k * c$ results of the sparse retriever.
Sub-graph Filtering: at subsequent iterations, a entities filter is added to ensure returned passages contain triples with entities which can be joined with the previous iteration's sub-graph.

3. Join Entities Synonym Expansion

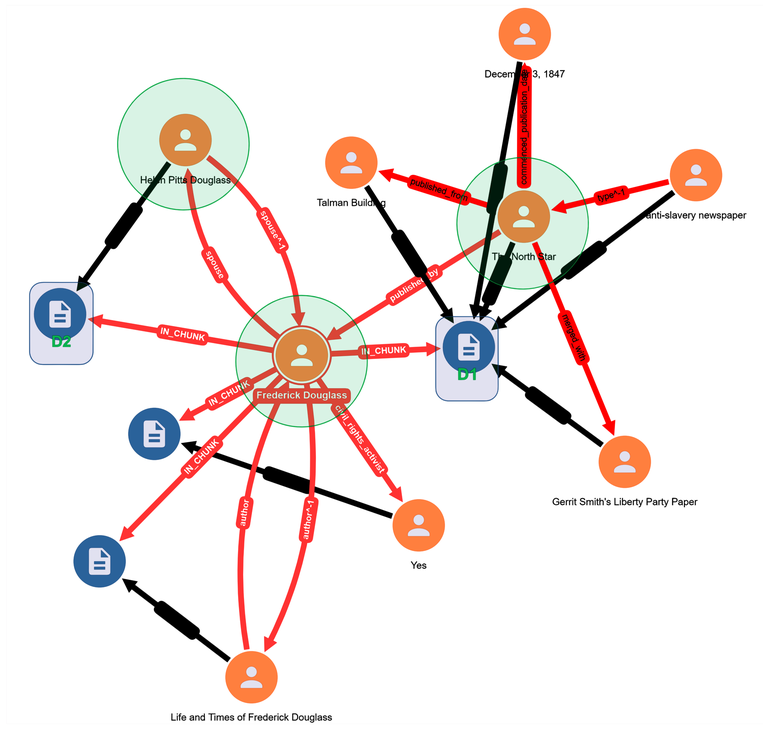
In the context where the graph is formed by text triplets, the aforementioned `subjects` or `objects` filter would miss cases where the join entity appears on different triples as different aliases. In order to address such cases and improve coverage, we consider the `same_as` index in order to expand the sub-graph of the joined entities with their synonyms.

4. Triples Shortlisting

The retrieved top-k passages may be connected to a large set of triples. As the retrieved top-k passages may be connected to a large number of triples, the `partial_triples` is used to shortlist the triples considered in the unsupervised tagger,since in the QA setting the whole triple is not expected to appear within an input natural language query.

5. Heuristic-based Query Re-writing

The highest-scoring entities using the vector-based matcher, 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.



D1: [score=0.925]: *The North Star* (anti-slavery newspaper). The North Star was a nineteenth - century *anti-slavery newspaper* published from the Talman Building in Rochester, New York *by abolitionist Frederick Douglass*. The paper commenced …

D2: [score=0.878]: *Helen Pitts Douglass* (1838–1903) was an American suffragist and abolitionist, known for being the *second wife of Frederick Douglass*. She also created the Frederick Douglass Memorial and Historical Association.
Associated Triple:

- {The North Star}-{published_by}-{Frederick Douglass}
- {Frederick Douglass}-{spouse}-{Helen Pitts Douglass}

[Q1] = "Who married the publisher of abolitionist newspaper The North Star?"
[Q2] = "Who married Fredrick Douglas?"
(The North Star)-[published_by]->(Frederick Douglass)-[spouse]->(Helen Pitts Douglass)

Figure: Example of an input query for GRiever.

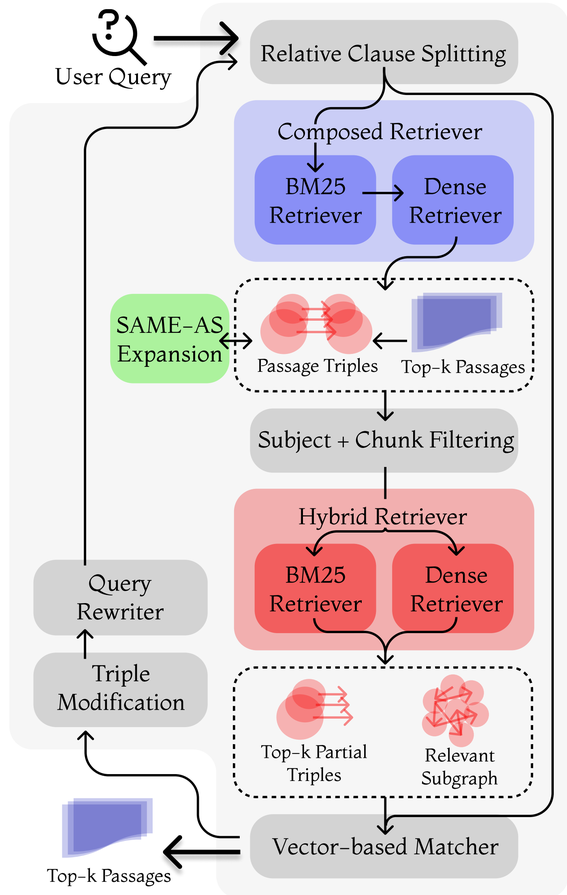


Figure: The architecture of GRiever.

Is GRiever efficient?

- GRiever does not require any LLM usage
- GRiever uses a lightweight vector-based entity matcher for query rewriting
- GRiever leverages efficient indexing structure and filtering to reduce latency
- GRiever has performance comparable to some agentic Graph-RAG frameworks, whilst maintaining runtime similar to dense base retriever on larger datasets

	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: Retrieval performance and runtime comparison.

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: End-to-end QA results.

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
wo/ `shortlisting`	0.452	0.532	0.574	0.594	0.695	0.713	0.739	0.870	0.916
wo/ `composed_retriever`	0.438	0.514	0.555	0.672	0.733	0.731	0.792	0.887	0.920

Table: Ablation study across different index configurations