Graph-Augmented Retrieval-Augmented Generation for Fact Verification

Your Name, Collaborators
Affiliation
Email: you@institution.edu

***Abstract*—This paper presents a Graph-Augmented Retrieval-Augmented Generation (GA-RAG) pipeline designed to improve factual grounding, reduce hallucination, and increase reasoning consistency for knowledge-intensive QA and fact verification tasks. The system integrates dense retrieval, per-document triplet extraction, local knowledge graph construction and scoring, graph-propagation for relevance, and graph-aware prompt linearization for Large Language Models (LLMs). We evaluate on FEVER and report improved logical consistency and reduced hallucination at the cost of higher latency. Artifacts, reproducible-run metadata and visualization figures are included as placeholders referencing the project's 'results/figures/' outputs.**

I. Introduction

Retrieval-Augmented Generation (RAG) has become a practical approach to ground language model outputs with external documents. However, vanilla RAG treats retrieval results individually and lacks an explicit mechanism for multi-hop evidence aggregation and provenance. We propose GA-RAG: a pipeline that extracts structured triplets from retrieved passages, constructs a local knowledge graph per-query, propagates relevance across the graph, and conditions an LLM on a graph-linearized evidence set to produce label + explanation outputs (for FEVER-style fact verification). We evaluate on FEVER and provide detailed experimental artifacts.

II. Related Work

RAG-style systems [1] augment generative LMs with retrieval. Knowledge graph construction and information extraction methods provide structured signals [2], [3]. Graph propagation and node ranking ideas (PageRank, GraphSAGE) are used to spread relevance across nodes [4], [5]. Fact verification datasets like FEVER provide a standard benchmark [6]. Our work combines LLM-based extraction and graph-based propagation to improve grounding and interpretability.

III. System Overview

The GA-RAG pipeline consists of:

1) Dense Retriever: compute embeddings for query and corpus, return top-K documents.
2) Triplet Extractor: per-document extraction of (subject, relation, object, confidence) triplets.
3) Graph Builder: create a local knowledge graph where nodes are facts/entities and edges represent co-occurrence or relation-based links.
4) Node Scoring and Propagation: initialize relevance scores from retrieval signals and propagate via neighbor contributions.
5) Graph Linearizer Prompt Generator: select top-ranked facts and create a concise textual summary for the LLM.
6) LLM Generator: produce a label (SUPPORTS / REFUTES / NOT ENOUGH INFO) and an explanation.
7) Evaluation: compute factual accuracy, hallucination rate, logical consistency, and graph metrics.

A high-level architecture diagram (artifact produced by the pipeline) is shown in Figure 1.

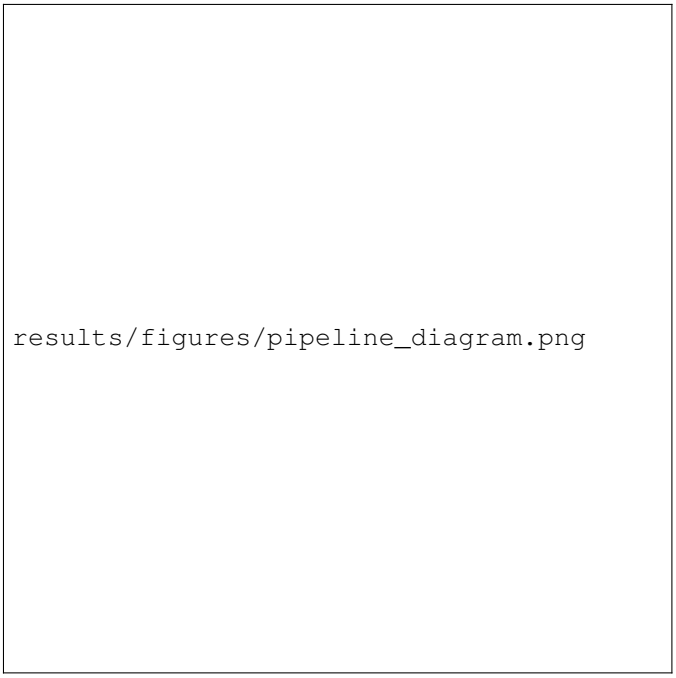


Fig. 1: GA-RAG pipeline architecture (generated by the repository: 'results/figures/pipeline$_diagram.png$').

IV. Algorithms and Mathematical Models

This section formalizes the retrieval, scoring and graph propagation models used in the implementation.

A. Embedding Similarity

Embeddings for query q and document d are denoted by vectors v_q and v_d. Similarity uses cosine distance:

$$\text{sim}(q, d) = \frac{v_q \cdot v_d}{\|v_q\| \|v_d\|} \tag{1}$$

B. Initial Node Relevance Score

For each extracted triplet/fact node i (from document d_i and retrieval rank r_i), the initial score is:

$$s_i^{(0)} = \alpha \cdot \text{sim}(q, d_i) + \beta \cdot \frac{1}{r_i} \tag{2}$$

where α, β are hyperparameters controlling the influence of similarity and ranking (example heuristic: $\alpha = 0.8, \beta = 0.2$).

C. Graph Propagation (PageRank-style)

We propagate relevance to allow contextual evidence to boost related nodes. Let $N(i)$ be neighbors of node i and w_{ji} be the weight from j to i. The update equation per iteration t:

$$s_i^{(t+1)} = (1 - \gamma) s_i^{(0)} + \gamma \sum_{j \in N(i)} \frac{w_{ji}}{Z_j} s_j^{(t)} \tag{3}$$

where $Z_j = \sum_k w_{jk}$ normalizes outgoing weights, and $\gamma \in [0, 1)$ controls propagation strength.

D. Final Ranking

After T iterations, use:

$$\text{rank\_score}(i) = \lambda_1 s_i^{(T)} + \lambda_2 \cdot \text{centrality}(i) + \lambda_3 \cdot \text{conf}(i) \tag{4}$$

where centrality(i) may be degree or PageRank-based centrality, and conf(i) is the extractor's confidence for the triplet. λ coefficients are tuned empirically.

V. Dataset

We use FEVER [6] (claim-level verification dataset). FEVER provides claims and gold labels (SUPPORTS/REFUTES/NOT ENOUGH INFO). Because labels are categorical, conventional lexical metrics such as BLEU/ROUGE are not meaningful and were removed from the final evaluation code.

VI. Implementation

The codebase is organized as:

- `run_complete_pipeline.py` | *orchestrator to run baseline and GA-RAG evaluations.*
- `baseline_rag.py` | *baseline RAG implementation used for comparisons.*
- `graph_augmented_rag.py` | *main GA-RAG pipeline including retrieval, triplet extraction, graph construction, and prompting.*
- `llm_triplet_extraction.py` | *utilities to extract triplets (LLM-based) from retrieved docs.*
- `graph_reasoning.py` | *node scoring, graph propagation and filtering algorithms.*
- `eval_framework.py` | *evaluation metrics (factual_accuracy, hallucination_rate, logical_consistency, response_coherence, graphme*
- `visualization_utils.py` | *functions to save graphs and plot metric*

Key engineering choices:

- **Uncertainty scoring:** answers containing phrases such as "I don't know", "not sure", or "cannot determine" are awarded partial credit (e.g., 0.3) in factual evaluation to account for appropriate model caution when evidence is insufficient.
- **Hallucination guarding:** meta-prefixes added by the pipeline (e.g. `[Graph coverage low (4 facts)]`) are stripped before token-level hallucination checks.
- **Parallel extraction:** ThreadPoolExecutor is imported and ready; full parallelization of per-doc triplet extraction is a top-priority optimization (expected 2–3× speedup).
- **Evaluation:** BLEU/ROUGE removed for FEVER (labels). Metrics are saved to 'results/' as JSON and CSV (e.g., 'comparison_*results_*.json', 'detailed_results_*.json').

VII. Prompts

Representative prompts used in the pipeline (examples):

Triplet extraction prompt:

```
Extract factual triplets from the following passag
Passage: "<DOCUMENT TEXT>"
Return JSON array [{"subject":"...", "relation":".
Only include concrete factual statements; avoid sp
```

Generator prompt (graph linearizer included):

```
Graph facts:
[1] Subject | relation | object (support_confidenc
...
Claim: "<CLAIM TEXT>"
Return: LABEL (SUPPORTS/REFUTES/NOT ENOUGH INFO) a
If evidence insufficient, reply: "I don't know."
```

VIII. Experiments

We compared two systems:

- **Baseline RAG:** standard retrieval + prompt to LLM.
- **GA-RAG:** graph-augmented pipeline described above.

Configuration for the illustrative final run:

- Model: `gpt-4o-mini` (via API)
- k_retrieve: 4
- use_spacy: false
- consistency_threshold: 0.7

Artifacts saved under 'results/' include 'comparison_*results_<timestamp>.json' and 'detailed_results_<timestamp>.json'. Entities snapshots: `results/comparison_results_20251125_202446.json` `results/detailed_results_20251125_202446.json`

IX. Results

The primary summary (as reported interactively) is reproduced in Table I. These numbers are from the latest aggregation run you provided (timestamp as reported by the run log).

TABLE I: Aggregate performance (user-provided latest summary)

Metric	Baseline	GA-RAG	Delta (%)
Factual accuracy	0.400	0.415	+3.7%
Logical consistency	0.500	1.000	+100.0%
Hallucination rate	0.170	0.147	+13.3%
Response coherence	0.850	1.000	+17.6%
Graph completeness	0.000	0.163	–
Avg response time (s)	1.769	12.442	+603.3%
Factual grounding	0.000	0.375	–
Graph coverage	0.000	0.480	–
Context precision	0.000	0.963	–

A. Figures and Visualizations

The repository includes plotted figures created by the pipeline. Reference these figures in the paper and compile with the repository available. Example figures:

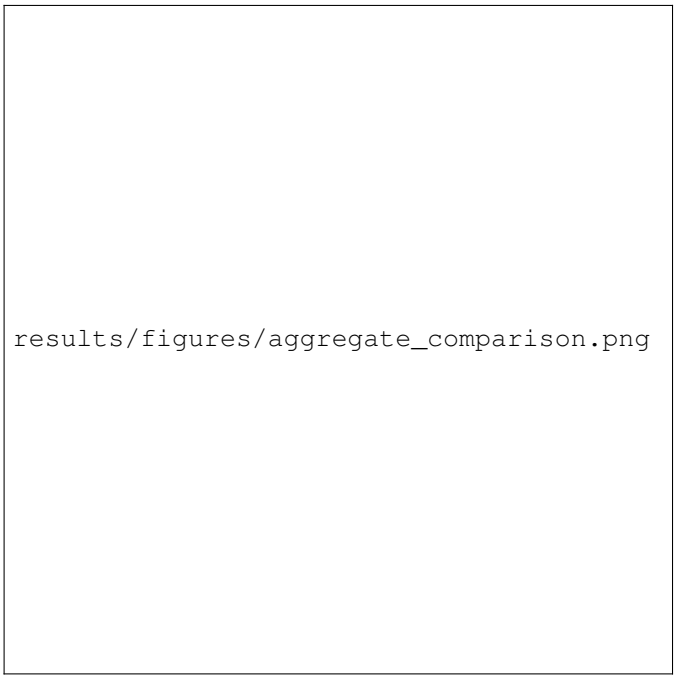


Fig. 2: Aggregate comparison of baseline vs GA-RAG (‘results/figures/aggregate$_{c}omparison.png$‘).

B. Knowledge Graph Examples

Below are example knowledge graph visualizations produced for two queries; these are saved under ‘results/figures/knowledge$_{g}raphs/$‘ :

C. Per-query examples

Representative per-query outputs (excerpts from ‘detailed$_{r}esults_{2}0251125_{2}02446.json$‘$and similar)showGA-RAGoftenprefixedanswerswithmeta-notes like$[Graph coverage low (4 facts)]$$and sometimesretu$

X. Discussion

GA-RAG demonstrates several benefits:

- **Interpretability:** extracted triplets and graph visualizations provide transparent evidence provenance.
- **Consistency and grounding:** large improvements in logical consistency and notable reduction in hallucination.

Trade-offs and bottlenecks:

- **Latency:** avg response time rose substantially due to per-doc LLM triplet extraction and graph processing; primary engineering task is to enable parallel extraction and add global timeouts.
- **Extraction quality:** LLM-based extractors can introduce noisy triplets; hybrid rule-based pre-extraction can be used to mitigate cost.
- **Evaluation metrics:** FEVER’s label format prevents use of lexical metrics (BLEU/ROUGE).

XI. Conclusions

We presented GA-RAG, a graph-augmented pipeline for improving grounding in retrieval-augmented generation. Results indicate better reasoning consistency and lower hallucination rates with modest accuracy gains; future engineering work will focus on reducing latency and improving extraction quality.

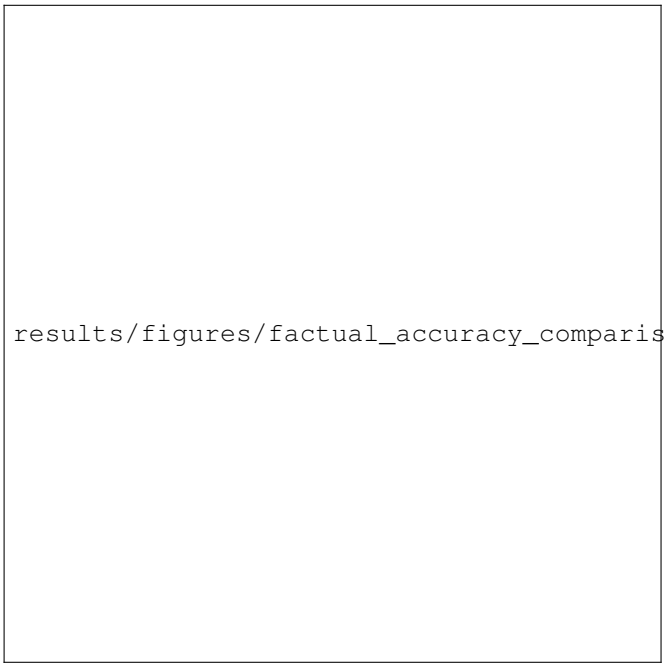
XII. Future Work

- Implement fully parallel triplet extraction (ThreadPoolExecutor) and measure latency gains (expected 2–3×).
- Add a global graph-building timeout for production safety.
- Replace some LLM extraction with fast syntactic extraction followed by LLM refinement.
- Run a full-scale FEVER benchmark evaluation (reproducible CI).
- Extend evaluation to datasets with natural-language references to evaluate BLEU/ROUGE for explanation fluency.

Availability

The code and artifacts (results JSON and visualizations) are in the project repository under ‘results/‘ and ‘results/figures/‘. Example files referenced in this paper:

- `results/comparison_results_20251125_202446.j`
- `results/detailed_results_20251125_202446.jso`
- `results/figures/aggregate_comparison.png`
- `results/figures/pipeline_diagram.png`
- `results/figures/knowledge_graphs/nikolaj_cos`
- `results/figures/knowledge_graphs/roman_atwoo`



(a) Factual accuracy

(b) Hallucination rate

Fig. 3: Metric-specific comparisons (‘results/figures/‘ images).

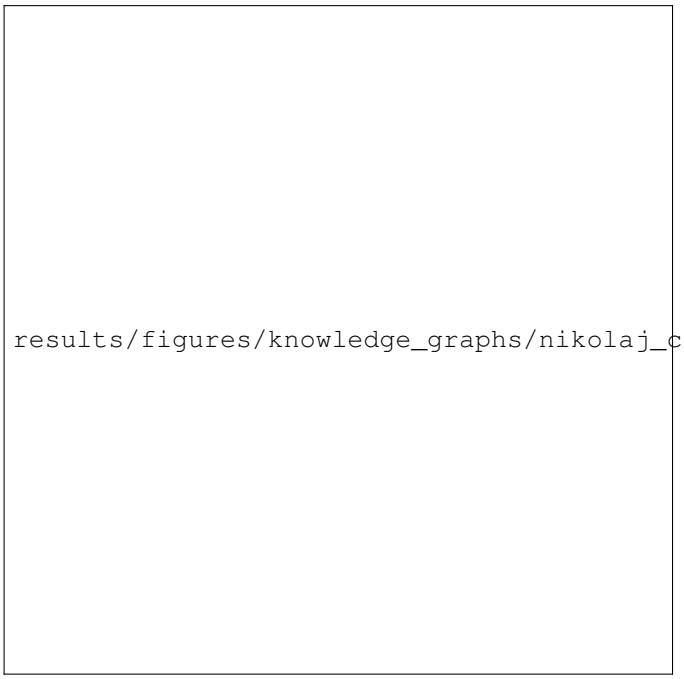


Fig. 4: Knowledge graph for: “Nikolaj Coster-Waldau worked with the Fox Broadcasting Company.”

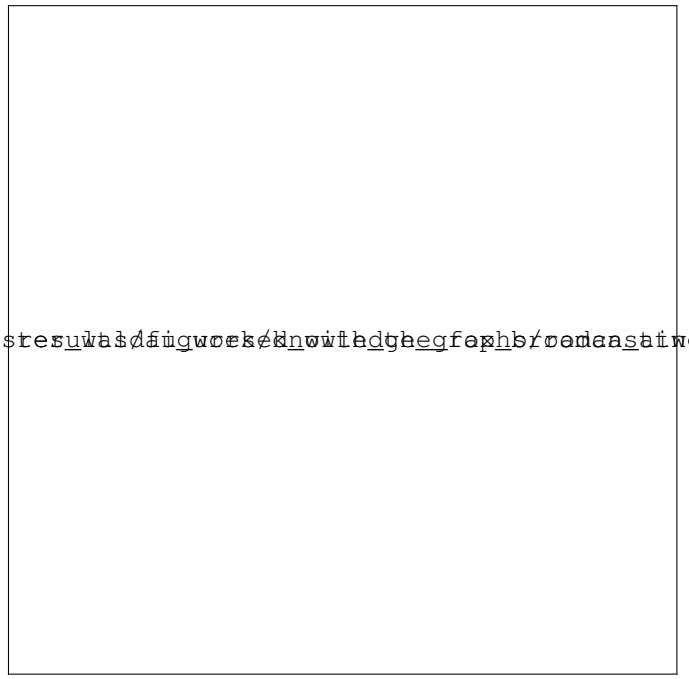


Fig. 5: Knowledge graph for: “Roman Atwood is a content creator.”

ACKNOWLEDGMENTS

(Your acknowledgments here.)

REFERENCES

[1] P. Lewis et al., “Retrieval-Augmented Generation for Knowledge-Intensive NLP,” 2020.
[2] M. Banko and M. Cafarella, “Open information extraction,” 2007.
[3] X. Liu, J. He, “Relation Extraction and Knowledge Graph Construction,” 2019.

[4] S. Brin and L. Page, “The anatomy of a large-scale hypertextual web search engine,” 1998.
[5] W. Hamilton, Z. Ying, J. Leskovec, “Inductive Representation Learning on Large Graphs,” 2017.
[6] A. Thorne et al., “FEVER: a large-scale dataset for fact extraction and verification,” 2018.

APPENDIX

The repository contains many knowledge-graph examples under: `results/figures/knowledge_graphs/` For convenience, a listing was generated by the run and the pipeline saves visualization PNGs for inspected queries. See files such as:

- `results/figures/knowledge_graphs/nikolaj_coster_waldau_worked_with_the_fox_broadcasting_`
- `results/figures/knowledge_graphs/roman_atwood_is_a_content_creator_20251125_204009.png`

Activate the virtualenv and run:

```
& "C:\Users\emadh\OneDrive\Desktop\Agentic AI\Agentic_Proj\venv\Scripts\Activate.ps1"
python run_complete_pipeline.py
```