Graph-Based Cache-Augmented Generation

Siyuan Zhang Zeqiao Huang
Yinfei Wang Libo Gu

Introduction

Retrieval-Augmented Generation (RAG) boosts LLMs with external knowledge but suffers from latency and selection errors. Cache-Augmented Generation (CAG) addresses this by preloading knowledge into the KV-cache. We explore graph-based cache strategies—relational summaries, JSON knowledge, and Microsoft Knowledge Graphs—finding that while they reduce context size and prep time, they can harm accuracy if misaligned or oversized. Properly sized, structured caches preserve performance, highlighting the trade-off between efficiency and accuracy in retrieval-free systems.

Methodology

Model: Llama-3.1-8B-Instruct **Data:** SQuAD (Stanford Question Answering Dataset)

Relational Summary Generation: Leveraged lightweight LLMs (ChatGPT-4o Mini, Gemini 2.5 Flash Lite) to create concise relational summaries, enhancing structural context of cached knowledge.

Json As Knowledge: Utilized LLMs to build knowledge graphs and convert them into Json format input for model cache.

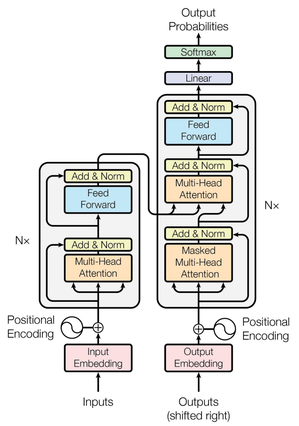
Experimental Setup: Conducted evaluations using Cache-Augmented Generation (CAG) across three scenarios (1/100, 3/500, 4/500), controlling variables like data scale, input configuration (Base, RelSum, Microsoft KG), and measuring response time and semantic similarity.

GraphRAG Integration:

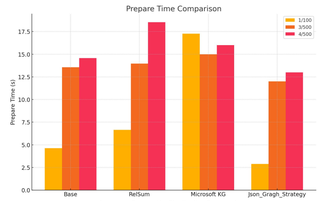
- **Vector Approaches**: Tested embedding-based methods (direct embeddings and fine-tuned projections) but found poor accuracy due to embedding information loss.
- **Textual Approach**: Integrated GraphRAG summaries as textual context, slightly improving accuracy (79% vs. 78%) and semantic consistency.

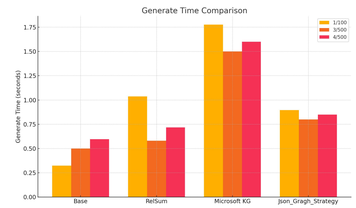
Evaluation Improvement: Replaced cosine similarity with Hugging Face instruction-tuned model for more accurate QA assessment, addressing paraphrasing, numerical format sensitivity, and semantic completeness.

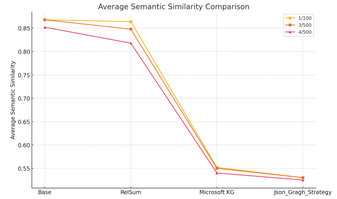
Experimental Variables Control	
Independent	• Configuration: Base, RelSum, Microsoft KG, Json_Graph_Strategy • Data scale: 1/100, 3/500, 4/500
Dependent	• Prepare time: Data load → preprocess (ms) • Generate time: Core execution (excl. I/O) • Similarity: BERT-base cosine (median of 3)
Constants	• Hardware: Docker (4C/16G) • Software: Py3.10.11/Torch2.0.1

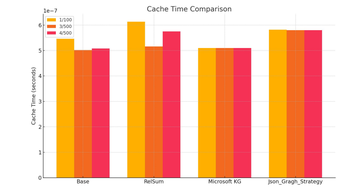


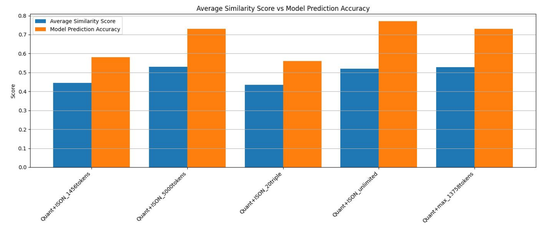
Experiment Result









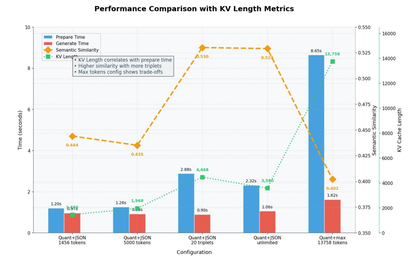


Analysis

Performance Analysis of Knowledge Strategies

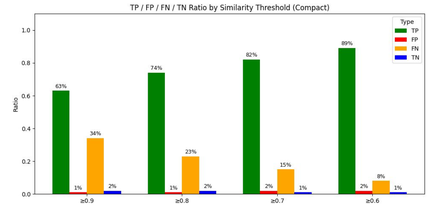
1. **JSON-as-Knowledge**
   - Semantic Similarity: **0.530**
   - Key Issues:
     - ▸ Missing critical information in flattened structures
     - ▸ Poor graph coherence
     - ▸ Model struggles with relational reasoning
   - Efficiency: **0.90s** generation time (slower than baseline)
2. **Microsoft Knowledge Graph (KG)**
   - Semantic Similarity: **0.552**
   - Strengths: Structured relations
   - Weaknesses: Shallow information depth
   - Efficiency: **17s** prep time (slowest), **1.78s** generation time
3. **Base CAG (Contextualized Attention Graph)**
   - Best Performance: Higher semantic similarity + optimal efficiency
   - Efficiency: **4.62s** prep time, **0.32s** generation time (fastest)

4、KV Cache Length Insights



- Optimal: Align KV cache length with input text length
- Mismatch Impact: Similarity drops to **0.402**
- Match Benefit: **25% accuracy gain**

5、Limitations of Embedding-Based Evaluation



- High FN Rate: Strict thresholds cause up to 34% FN.
- Low TP Rate: Rarely identifies correct outputs as correct.

Conclusion

Limitations:

- Summarizing the full knowledge graph into context was highly lossy.
- Truncating to text or encoding as a static embedding omitted key information.
- Resulted in only marginal accuracy gains.

Future Work:

- Use dynamic, query-aware graph integration to retrieve relevant subgraphs per query.
- Explore hybrid methods combining textual summaries with vector-based graph representations for richer context.

https://github.com/Jasonzzzz28/CAG