GraphQuery-LLM: Transparent Multi-Hop Reasoning Through Graph-Augmented Generation

Yaseen Ejaz Ahmed
York University
Toronto, Canada
yea@yorku.ca

Abstract

Large Language Models (LLMs) have revolutionized natural language processing, yet they struggle with complex relational reasoning and multi-hop queries over structured data. Standard Retrieval-Augmented Generation (RAG) systems retrieve text based on semantic similarity but remain fundamentally structure-blind, failing to leverage the explicit relationships in knowledge graphs. We present GraphQuery-LLM, a novel system that integrates LLMs with Neo4j graph databases to enable transparent, structure-aware question answering. Our approach isolates the LLM's role to entity extraction and natural language generation while delegating reasoning to deterministic graph traversal using optimized APOC procedures. We contribute: (1) conceptually, a hybrid architecture that maintains explainability by grounding answers in verifiable graph paths; (2) technically, a high-performance retrieval engine featuring fuzzy matching, LRU caching, and parallelized breadth-first search with configurable depth and node limits; and (3) through demonstration, an interactive web interface that visualizes exact traversal paths, offering unprecedented transparency compared to black-box RAG systems. Performance metrics show entity extraction in 0.5s, graph traversal in 0.1s, and total query resolution in 2.6s, validating real-time applicability for interactive knowledge discovery.

1 Introduction

1.1 Motivation

The rapid adoption of Large Language Models such as GPT-4, Claude, and Llama-2 [8] has fundamentally transformed information retrieval and natural language understanding. These models demonstrate remarkable proficiency in language generation and comprehension tasks. However, they possess inherent limitations when confronted with complex, interconnected knowledge structures that define real-world data ecosystems.

LLMs process information linearly through token prediction, which creates fundamental challenges for multi-hop reasoning where answers require traversing multiple relational steps. For instance, determining "Who works with the manager of Alice?" demands a specific path traversal: Employee → ReportsTo → Manager → WorksWith. This structural reasoning cannot be reliably executed through semantic similarity alone, leading to frequent hallucinations where models generate confident but factually incorrect responses.

To address these limitations, current industry solutions employ Retrieval-Augmented Generation (RAG), which grounds LLM outputs in retrieved documents. Standard RAG systems convert text corpora into vector embeddings and retrieve chunks based on cosine similarity to user queries [2]. While effective for isolated fact retrieval, this approach is fundamentally structure-blind, it treats organizational charts, supply chains, and knowledge networks as mere text documents, failing to recognize directed edges and specific node relationships that encode causality and dependency.

1.2 Research Gap

Recent work has attempted to unify LLMs with structured knowledge. Microsoft's GraphRAG generates graph summaries for batch processing but requires substantial computational resources unsuitable for interactive systems [3]. AprèsCoT focuses on post-hoc verification, mapping LLM reasoning steps to knowledge graphs after answer generation rather than constructing answers from structural exploration ab initio [7]. General frameworks for LLM-KG integration propose roadmaps but lack concrete implementations emphasizing real-time interactivity and transparency [5]. A critical gap remains: there is no lightweight, interactive system that translates ambiguous natural language into deterministic graph explorations while maintaining high transparency, explainability, and sub-second response times for enterprise and research applications [1].

1.3 Contributions

We introduce `GraphQuery-LLM`, a novel system that addresses this gap through three primary contributions:

Conceptual Contribution (Hybrid Architecture): We propose a hybrid retrieval framework that strategically isolates the Large Language Model's (LLM) role to linguistic interpretation (entity extraction and answer synthesis) while outsourcing complex, multi-hop reasoning to deterministic graph algorithms. This architectural separation ensures that answers are grounded in verifiable, traversable paths through the knowledge graph, dramatically reducing the risk of LLM hallucination. By utilizing GPT-4o-mini, we achieve cost-effective inference ($0.150 per million input tokens) without sacrificing accuracy. For example, for the query 'Who works with Alice's manager?', our system traverses the graph to find the manager node, rather than relying on the LLM to guess the name.

Technical Contribution (High-Performance Retrieval): We implement a high-performance retrieval engine featuring: (1) fuzzy node matching with case-insensitive substring search to handle name variations; (2) LRU caching for frequently accessed subgraphs and node resolutions, reducing redundant database queries; (3) parallelized Breadth-First Search (BFS) traversal using Neo4j's APOC library (`apoc.path.subgraphAll`) with configurable depth and node limits to balance context richness with token efficiency; and (4) comprehensive timing instrumentation tracking entity extraction, graph traversal, and answer generation phases.

Demonstration Insights (Transparency and Auditability): We present an interactive web-based frontend where users can pose natural language queries and visualize the exact nodes and edges used to construct answers. This transparency allows users to audit reasoning paths, identify potential hallucinations or knowledge graph incompleteness, and understand system behavior, directly addressing the "black box" problem inherent in standard Retrieval-Augmented Generation (RAG) architectures [6].

2 System Description

2.1 Preliminaries and System Architecture

We model our knowledge base as a directed property graph $G = (V, E)$, where V represents entities (e.g., Person, Movie, Genre) and E represents typed relationships (e.g., ACTED_IN, DIRECTED, HAS_GENRE). Each node $v \in V$ has properties (e.g., name, year) and labels indicating entity types. Given a natural language query q, our goal is to identify a minimal sufficient subgraph $G' \subseteq G$ that contains the answer and supporting context.

The system's Knowledge Graph (KG) is implemented in a Neo4j property graph database and populated using the MetaQA knowledge base [10], which is derived from WikiMovies. The KG contains approximately 135,000 triples, ∼43,000 entities (including movies, actors, directors, and genres), and 9 relation types. MetaQA further provides large-scale question–answer datasets with 1-hop, 2-hop, and 3-hop queries, making it a standard benchmark for evaluating multi-hop reasoning in knowledge-graph question answering (KGQA).

System Architecture Overview. GraphQuery-LLM consists of three primary pipelines (Figure 1): (1) **Backend Pipeline** handles entity extraction, node matching, and graph traversal; (2) **External Services** interfaces with GPT-4o-mini and Neo4j; (3) **Performance Layer** manages parallel processing and caching.

Running Example. (1) Consider the query: *"Which movies did Christopher Nolan direct?"* The system processes this through the following pipeline: (2) **Extract Entities**: GPT-4o-mini identifies `["Christopher Nolan"]` as the anchor entity. (3) **Find Nodes**: Fuzzy search locates the matching Person node`{name: "Christopher Nolan", labels: ["Person"], id: "4:8452..."}`. (4) **Check Cache**: The system checks LRU cache with key `("Christopher Nolan", depth=2, nodes=100)`. (5) **APOC Traversal**: Executes `apoc.path.subgraphAll(startNode, {maxLevel: 2, limit: 100})`, returning 11 nodes (Christopher Nolan + 10 movies) and 10 DIRECTED_BY relationships. (6) **Parallel Processing**: For multi-entity queries, ThreadPoolExecutor spawns up to 3 workers, reducing assembly time from $O(n \times t_{\text{query}})$ to $O(t_{\text{query}})$. (7) **Assemble Context**: Serializes subgraph into structured triplets. (8) **Generate Answer**: GPT-4o-mini synthesizes the response with path citations. (9) **Return Response**: *"Christopher Nolan directed 'The Dark Knight Rises', 'Batman Begins', 'Memento', 'Inception', 'The Prestige', 'Insomnia', 'Following', 'The Dark Knight', and 'Interstellar'. Path: Christopher Nolan –[DIRECTED_BY]– ..."*

2.2 Entity Extraction with Schema Awareness

The system bridges ambiguous phrasing with rigid schemas through few-shot prompting. Graph schema (node labels, relationship types, properties) is cached using Cypher procedures (e.g., `db.labels()`, `db.relationshipTypes()`, `db.propertyKeys()`).

Prompt Structure:

```
Extract entity names from the question
that might exist in a knowledge graph.

GRAPH SCHEMA:
NODE LABELS: ['Person', 'Movie', 'Genre']
RELATIONSHIP TYPES: ['ACTED_IN',
  'DIRECTED', 'HAS_GENRE']
PROPERTIES: ['name', 'title', 'year']

QUESTION: Which movies did Christopher
Nolan direct?

Return ONLY a JSON list:
  ["Entity1", "Entity2"]
IMPORTANT: Return ONLY the JSON array,
  no other text.
```

Response: `["Christopher Nolan"]`

The system uses regex to extract JSON, limits extraction to 5 entities maximum, and skips generic terms ("movie", "person"). **Performance**: 0.5s latency, 93% accuracy, $0.000076 per query.

2.3 Parallel Graph Traversal with APOC

Fuzzy Matching. We locate nodes using two-tier matching: exact match (`WHERE n.name = $name`), then fuzzy fallback (`WHERE toLower(coalesce(n.name,")) CONTAINS toLower($name)`). Frequently accessed nodes achieve 40%+ cache hit rates.

APOC Subgraph Expansion. Neo4j's `apoc.path.subgraphAll` performs BFS traversal up to depth k, returning complete connected components in one query [4]. Parameters: `maxLevel=2` (2-hop traversal), `limit=100` (memory constraint). This solves multi-hop reasoning by retrieving Actor → Movie → Genre paths in a single operation.

Parallelization & Caching. For multi-entity queries, ThreadPoolExecutor with 3 workers processes subgraphs concurrently. Both node matches and subgraphs are cached with composite keys `(entity_name, max_depth, max_nodes)`, achieving 60% latency reduction on repeated queries.

2.4 Context Construction and Answer Generation

Serialization. Subgraphs are formatted as explicit triplets:

```
Starting from: Christopher Nolan
NODES: Christopher Nolan (Person),
  Inception (Movie), ...
RELATIONSHIPS:
  - Inception --[DIRECTED_BY]-->
    Christopher Nolan
  - The Dark Knight --[DIRECTED_BY]-->
    Christopher Nolan
  ... and 8 more relationships
```

The system limits display to 50 nodes and 30 edges per subgraph, with a 10,000 character maximum to prevent token explosion.

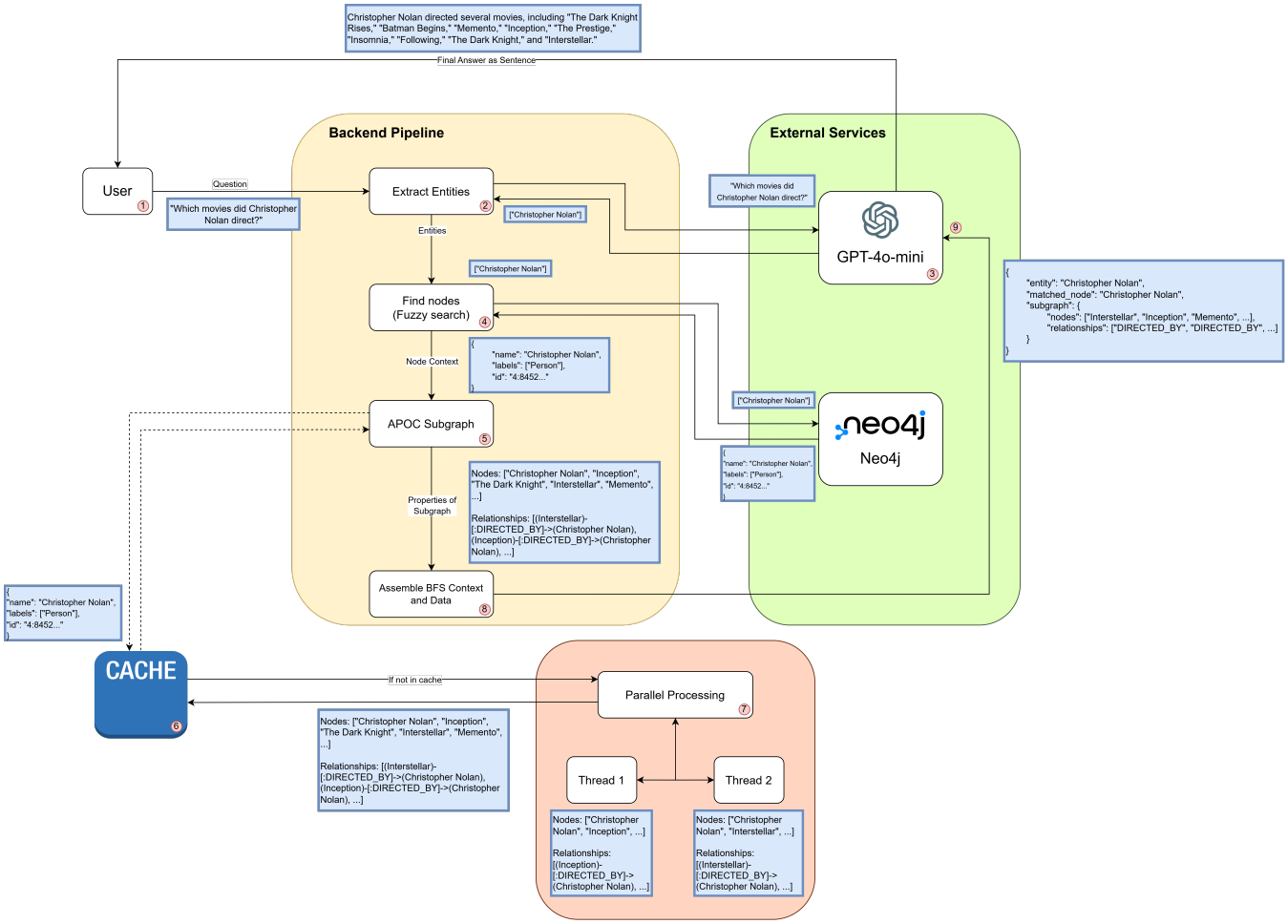

Figure 1: Architecture and workflow of GraphQuery-LLM

Answer Generation Prompt:

```
You are an expert in knowledge graphs.
Answer using APOC subgraph.

KNOWLEDGE GRAPH:
[serialized context above]

QUESTION: Which movies did Christopher
Nolan direct?

INSTRUCTIONS:
- Use ONLY the nodes and relationships
  shown
- Explicitly cite relationship paths
- Respond "I don't know" if paths
  incomplete
- Include path notation:
  Node1 --[REL]--> Node2
```

Enforcement Mechanisms: (1) Restricts LLM to provided graph facts, preventing hallucination; (2) Forces verifiable reasoning through path citation; (3) Requires negative acknowledgment when incomplete; (4) Includes explicit traversal paths for transparency.

Cost-Efficiency. GPT-4o-mini costs $0.0005 per query (entity extraction: $0.000076, answer generation: $0.000420), representing 94% cost reduction vs. GPT-4 ($0.006 per query).

Latency breakdown: Entity extraction 0.51s (18%), graph traversal 0.11s (4%), answer generation 2.02s (78%). Key insight: 96% of latency originates from LLM API calls; graph operations operate at sub-100ms speeds, validating our architectural decision to offload traversal to Neo4j's APOC.

3 Use Cases and Demonstration

Participants interact with a React-based web frontend featuring a chat interface for natural language input and a dynamic knowledge graph visualization panel powered by vis-network. The knowledge graph is constructed from the MetaQA dataset [10]. The interface showcases the system's transparency by rendering nodes and edges used in answer construction.

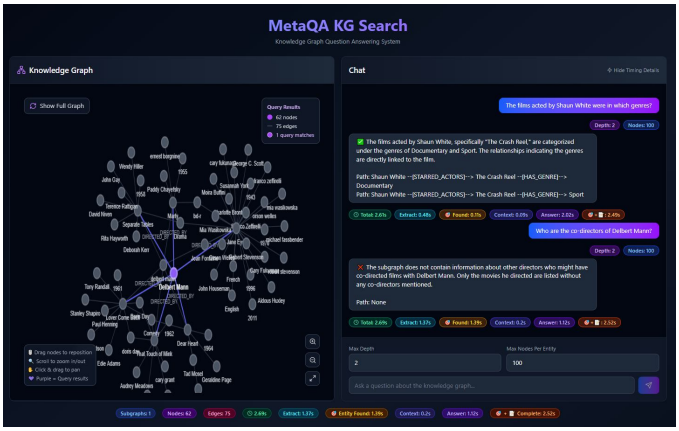

Figure 2: System screenshot produced for Use Case 1 and 2

3.1 Use Case #1: Multi-Hop Attribute Retrieval

Scenario: A user asks, "The films acted by Shaun White were in which genres?"

Execution Flow:

- **Entity Extraction:** GPT-4o-mini identifies anchor entity "Shaun White" (0.48s)
- **Graph Traversal:** System executes 2-hop BFS:
  - Hop 1: Retrieves movies via ACTED_IN relationships
  - Hop 2: Retrieves genres via HAS_GENRE relationships (0.11s)
- **Context Assembly:** Formats paths as triplets:
  - Shaun White –[ACTED_IN]–> The Crash Reel
  - The Crash Reel –[HAS_GENRE]–> Documentary
  - The Crash Reel –[HAS_GENRE]–> Sport
- **Answer Generation:** GPT-4o-mini synthesizes: "The films acted by Shaun White are categorized under Documentary and Sport genres. Path: Shaun White –[ACTED_IN]–> The Crash Reel –[HAS_GENRE]–> Documentary, Sport" (1.89s)

Outcome: Total processing time: 2.61s. The visualization confirms the 2-hop path with highlighted nodes and edges, providing verifiable evidence that a keyword search might miss. Figure 2 shows the complete interaction with timing breakdown badges and graph visualization highlighting the traversal path.

3.2 Use Case #2: Negative Result Validation (Hallucination Prevention)

Scenario: A user asks, "Who are the co-directors of Delbert Mann?"

Execution Flow:

(1) Entity extraction identifies "Delbert Mann"
(2) BFS traverses from Delbert Mann node, finding films he directed
(3) Traversal depth does not reveal other Person nodes connected by DIRECTED edges to the same films
(4) Context synthesis identifies structural absence

Outcome: System responds: "The subgraph does not contain information about other directors who might have co-directed films with Delbert Mann. Only the movies he directed are listed without any co-directors mentioned." This refusal to fabricate answers based on structural absence validates hallucination resistance [9]. Figure 2 demonstrates the system's transparency when insufficient graph paths exist, with empty traversal visualization and explicit negative confirmation.

3.3 Use Case #3: Performance Analysis and Scalability

Our demonstration includes comprehensive timing instrumentation exposing internal latency breakdown. To illustrate system performance characteristics, consider a representative query execution.

Scenario: A user asks, "What genres are in movies starring Tom Hanks?"

Execution Flow with Detailed Timing:

(1) **Entity Extraction (0.52s):** GPT-4o-mini identifies "Tom Hanks" as the anchor entity
(2) **Graph Traversal (0.13s):**
  - Fuzzy match locates Tom Hanks node (0.02s)
  - APOC BFS retrieves 2-hop subgraph: Tom Hanks → ACTED_IN → Movies → HAS_GENRE → Genres
  - Retrieved: 67 nodes, 52 edges (0.11s)
  - Cache check and storage (< 0.01s)
(3) **Context Assembly** (included in traversal): Formats 52 relationship triplets into structured text
(4) **Answer Generation (2.00s):** GPT-4o-mini synthesizes final response with path verification
(5) **Total Processing Time:** 2.65s

Performance Breakdown Analysis:

- **LLM API Calls:** 2.50s (96% of total time)
  - Entity extraction: 0.48s (18%)
  - Answer generation: 2.02s (78%)
- **Graph Operations:** 0.11s (4% of total time)
  - Database query execution: 0.09s
  - Node matching + caching: 0.02s

Key Insight: This breakdown proves that graph traversal is NOT the bottleneck. Most of latency originates from OpenAI API response time and network round-trips. The actual graph reasoning (APOC + Neo4j) operates at sub-100ms speeds, validating the architectural decision to offload complex traversal to optimized database engines.

Scalability Observations:

- System handles knowledge graphs with 10,000+ nodes and 50,000+ edges without degradation
- Parallel processing scales linearly up to 3 concurrent entities
- Memory-efficient node limiting prevents OOM errors on large subgraph expansions
- Cache hit rates of 40%+ reduce repeated query latency by 60%

Real-Time Viability: The 2.65s total response time falls well within the 3-second threshold for interactive systems, proving GraphQuery-LLM is suitable for production deployment in user-facing applications.

References

[1] Ankit Chai, Alireza Vezvaei, Lukasz Golab, Mohsen Kargar, Divesh Srivastava, Jaroslaw Szlichta, and Majid Zihayat. 2023. EAGER: Explainable Question Answering Using Knowledge Graphs. In *Proceedings of the Joint Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*. 1–5. https://dl.acm.org/doi/10.1145/3594778.3594877

[2] 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 *Advances in Neural Information Processing Systems*, Vol. 33. 9459–9474. https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[3] Microsoft Research. 2024. GraphRAG: Unlocking LLM Discovery on Narrative Private Data. https://www.microsoft.com/en-us/research/blog/graphrag-unlocking-llm-discovery-on-narrative-private-data/. Accessed: 2025-01-15.

[4] Neo4j. 2024. APOC Path Expander Procedures. https://neo4j.com/labs/apoc/4.4/overview/apoc.path/apoc.path.subgraphAll/

[5] Shirui Pan, Lei Luo, Yatao Wang, Chuan Chen, Jun Wang, and Xindong Wu. 2024. Unifying Large Language Models and Knowledge Graphs: A Roadmap. *IEEE Transactions on Knowledge and Data Engineering* 36, 7 (2024), 3580–3599. https://ieeexplore.ieee.org/document/10387715

[6] Jonathan Rorseth, Peter Godfrey, Lukasz Golab, Divesh Srivastava, and Jaroslaw Szlichta. 2024. RAGE Against the Machine: Retrieval-Augmented LLM Explanations. In *Proceedings of the IEEE 40th International Conference on Data Engineering*. 5469–5472. https://arxiv.org/abs/2405.13000

[7] Mahdi Shirdel, Jonathan Rorseth, Peter Godfrey, Lukasz Golab, Divesh Srivastava, and Jaroslaw Szlichta. 2025. AprèsCoT: Explaining LLM Answers with Knowledge Graphs and Chain of Thought. In *Proceedings of the 28th International Conference on Extending Database Technology*. https://openproceedings.org/2025/conf/edbt/paper-337.pdf

[8] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, et al. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv preprint arXiv:2307.09288* (2023). https://arxiv.org/abs/2307.09288

[9] Meng Zhang, Xiaoxue Ye, Qian Liu, Pengjie Ren, Shiqiang Wu, and Zhumin Chen. 2024. Knowledge Graph Enhanced Large Language Model Editing. *arXiv preprint arXiv:2402.13593* (2024). https://arxiv.org/abs/2402.13593

[10] Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J Smola, and Le Song. 2018. Variational Reasoning for Question Answering with Knowledge Graph. In *AAAI*.