DrugRAG: Neo4j-Backed Graph-Augmented Retrieval for Drug-Interaction Knowledge

Karim Mohamed, Mohammed Yousef, Omar Morshdy
Computer Science and Engineering Department, Nile University, Cairo, Egypt
{k.mohammed2139, m.abdelnaser2107, o.ayman2149}@nu.edu.eg

***Abstract*—Large Language Models (LLMs) show promise in medical domains but struggle with accurate drug-interaction information retrieval due to knowledge cutoff and hallucinations. Retrieval-Augmented Generation (RAG) can inject external knowledge, yet typical text-retrieval RAG may miss complex relationships among drugs and effects. We introduce DrugRAG, a graph-augmented retrieval system that leverages a Neo4j knowledge graph of drug interactions to enhance LLM responses. Our methodology ingests two medical textbooks into a graph (3,296 nodes, 6,368 relationships) and uses an LLM to generate Cypher queries, retrieving pertinent subgraphs which are then provided to a 70B LLM for answer synthesis. In evaluations on 100 expert-crafted medical QA queries, DrugRAG outperformed baselines (LLM with no retrieval, standard RAG, and an agent-based RAG) in both semantic similarity to reference answers and factual correctness/clarity scores (judged by a GPT-4.1-based evaluator). The results show a ∼30% improvement in answer quality over no retrieval and ∼15% over text-RAG baselines. We also find that the larger Llama-3.3-70B-Instruct model yields significantly better answers than a smaller 3.1-8B model. We discuss the system design, experimental findings, and how graph augmentation improves reliability. All data sources are textbook-based and anonymized, ensuring compliance with privacy standards.**

I. Introduction

Large Language Models have demonstrated impressive abilities in generating human-like medical advice and explanations, but ensuring factual accuracy and up-to-date knowledge remains a challenge [3][4]. In critical domains like drug interactions, hallucinations or outdated information can have dangerous consequences. Retrieval-Augmented Generation (RAG) [1] addresses this by equipping LLMs with external knowledge retrieval, allowing models to ground their responses in relevant data. While most RAG systems use unstructured text corpora or vector databases for retrieval, complex medical queries often demand understanding of structured relationships (e.g., drug–drug interactions, side-effect causality) that are naturally represented in a *knowledge graph*. Knowledge graphs have been used to integrate biomedical information [5] and can enhance reasoning by explicitly linking related entities.

In this work, we propose **DrugRAG**, a Neo4j-backed graph-augmented retrieval system tailored for drug-interaction knowledge. Our approach extends the RAG paradigm by constructing a specialized medical knowledge graph from authoritative textbooks and using LLMs to query and reason over this graph. By translating user questions into Cypher graph queries, our system can precisely retrieve relevant interconnected facts (e.g. which drugs interact and what side effects result) that a traditional text-based RAG might miss or have to infer implicitly. We then feed the retrieved subgraph information into a powerful LLM to generate a comprehensive answer.

Our contributions are as follows. First, we design an end-to-end pipeline that combines heuristic medical entity extraction, a Neo4j graph database, and large LLMs (8B and 70B parameters) for query understanding and answer synthesis. Second, we demonstrate through experiments on 100 expert-written QA pairs that DrugRAG significantly improves answer correctness and clarity compared to baselines including no retrieval, standard embedding-based RAG, and an agentic multi-step RAG approach. Third, we show that leveraging a knowledge graph enables more accurate multi-hop reasoning (e.g., identifying a chain of drug interactions causing a symptom) with high precision. Finally, we address ethical and privacy considerations, using only public textbook data and ensuring our system is compliant with GDPR by design. Overall, DrugRAG illustrates the promise of graph-augmented retrieval in making LLM responses in healthcare more reliable and transparent.

II. Related Work

LLMs in Medicine and RAG: The medical community has begun exploring large LLMs for clinical applications [3], but reliability is a concern. Retrieval-Augmented Generation (RAG) [1] is a technique to ground LLM outputs in external knowledge. In a recent perspective, Yang *et al.* highlight that RAG can improve factual accuracy in healthcare by providing up-to-date context [4]. Standard RAG systems typically use vector similarity search over text documents. While effective for many tasks, this approach can struggle with complex queries that require synthesizing relationships across multiple documents or reasoning over structured facts.

Knowledge Graphs for Medical QA: Knowledge graphs (KGs) offer a structured representation of medical knowledge, capturing entities (e.g., drugs, symptoms) and relationships (e.g., *interacts_with*, *causes*) explicitly. Prior works have integrated KGs into QA and generation. Wu *et al.* proposed *MedGraphRAG*, a graph-based RAG framework that links user queries to credible medical sources via a triple-based graph, improving evidence-based answers [8]. Zhao *et al.*'s *MedRAG* leverages a hierarchical medical KG to enhance differential diagnosis generation [9]. Guo *et al.* introduce *LAB-KG*, which conditions an LLM on a knowledge graph of lab tests to produce more accurate interpretations [11]. These efforts demonstrate that KGs can improve generation specificity and traceability in the medical domain. Our work similarly uses a KG (focused on drug interactions) but distinguishes itself by employing an LLM to dynamically query the graph with Cypher, rather than a fixed retrieval pipeline.

Agentic and Multi-step RAG: Traditional RAG systems perform a single retrieval step. Recent research explores *Agentic RAG* [2], where autonomous agents (powered by LLMs) manage multi-step retrieval and reasoning. Such agents can plan queries, invoke tools, and refine search results iteratively, enabling more complex question answering. For example, an agentic approach might break a question into sub-queries and retrieve intermediate information before final answer synthesis. In our baselines we include an *Agentic RAG* variant, where an LLM performs two iterations of retrieval (with query refinement) on a text corpus. While agentic strategies add flexibility [2], they may still rely on unstructured data. By contrast, DrugRAG's use of a structured graph inherently supports multi-hop reasoning (the graph query can traverse multiple hops in one step), simplifying what an agent might achieve through multiple calls.

Natural Language to Graph Queries: Converting natural language queries into structured graph database queries is a challenging task that has gained attention. Feng *et al.* [13] harnessed LLMs with chain-of-prompt techniques to translate user questions into Cypher queries for knowledge base QA, showing that even complex graph queries can be generated with few-shot prompting. Hornsteiner *et al.* [12] developed a real-time text-to-Cypher interface using ChatGPT, demonstrating the feasibility of LLM-powered Cypher generation for Neo4j. Our methodology builds on these insights by prompting a smaller LLM (8B) to produce Cypher queries targeting our Neo4j drug graph. Unlike prior NL2Cypher studies focusing on general or enterprise data, we tailor prompts to the medical domain and incorporate domain heuristics (e.g., ensuring recognized drug names appear in the query). Additionally, we handle cases where the LLM query yields no results by simple re-prompting or relaxing query constraints (an approach similar to the error-handling strategy noted by Feng *et al.* [13]).

Neo4j in Healthcare: Graph databases like Neo4j have been utilized in healthcare research for data integration and analysis. Soni *et al.* [7] demonstrated Neo4j's utility in linking disparate data (Twitter discussions and medical knowledge) to study public opioid use patterns. Neo4j has also been used to model patient records and biomedical knowledge networks in various settings due to its flexibility and query capabilities. Our work uses Neo4j as the backbone for the drug-interaction knowledge graph, exploiting its Cypher query language to retrieve subgraphs relevant to a question. This approach benefits from Neo4j's ability to store and query relationships efficiently, something not directly possible with pure text retrieval. By integrating Neo4j with LLMs, we join a line of research showing that symbolic knowledge (graphs) and neural language models can complement each other [14][15].

MedGraphRAG Wu et al. (2024) propose **MedGraphRAG**, a Graph-RAG framework tailored to the medical domain that systematically integrates triple-linked KG construction with "U-Retrieval"—a combination of top-down precise retrieval and bottom-up response refinement. Their triple graph connects user documents to authoritative medical sources and controlled vocabularies, enabling LLMs to retrieve both structured KG facts and relevant text. MedGraphRAG was validated on nine medical QA benchmarks, two health fact-checking datasets, and a long-form generation set, consistently outperforming existing RAG and GraphRAG methods while ensuring grounded, evidence-based responses.

In summary, DrugRAG intersects multiple research threads: LLM grounding via retrieval, knowledge graph-based medical QA, and NL-to-database query generation. To our knowledge, it is one of the first systems to apply an LLM-driven Cypher query mechanism to a medical knowledge graph for enhanced retrieval. Next, we describe our methodology and system design in detail.

III. Methodology

Our DrugRAG pipeline comprises several stages: (1) knowledge ingestion from medical textbooks into a graph, (2) question understanding and graph query generation via an LLM, (3) graph-based retrieval of relevant facts, and (4) answer synthesis with a large LLM using the retrieved subgraph.

A. Stage 1: Knowledge Ingestion

We begin by constructing a specialized knowledge graph of drug interactions. We selected two authoritative medical textbooks focusing on pharmacology and drug interactions. The full text (chapters and appendices) was segmented into chunks of approximately 1,000 tokens each, to facilitate downstream processing and to serve as potential context pieces. We applied a heuristic Named Entity Recognition (NER) process to each chunk to identify key medical entities: *drug names*, *chemical compounds*, and *side-effects*. The NER was implemented using a combination of a dictionary (compiled from DrugBank [6] for known drug and compound names) and rule-based patterns (to capture variations in terminology).

When an interaction or side-effect relationship was mentioned in the text (e.g., "Drug A may increase the effect of Drug B, causing severe hypotension"), we extracted that relationship. Specifically, we define two primary relation types in our knowledge graph: **interacts_with** (between two drug entities) and **causes_side_effect** (linking a drug to a side-effect entity). Each extracted triple is added as an edge in the graph, connecting the corresponding nodes. If an entity was not already present, a new node was created. In cases where the same relationship was stated in multiple places, we increment a count or attach a source reference attribute, though for retrieval purposes we treat multiple mentions as one edge.

After ingestion, we obtained a knowledge graph with 3,296 unique nodes and 6,368 relationships. The node set includes hundreds of drugs (both generic and brand names), various compounds (active ingredients or chemical substance names), and dozens of distinct side-effects or adverse outcomes. Relationships in the graph encode known interaction pairs (drug-drug interactions) as well as drug-side effect links. Table I summarizes the graph's key statistics. The graph is stored in a Neo4j database (version 5.27.0) and can be visualized or queried via Cypher. Notably, the graph's structure allows retrieval of multi-hop connections (e.g., drug A interacts with drug B which causes side effect X) through a single query, an advantage over flat text retrieval.

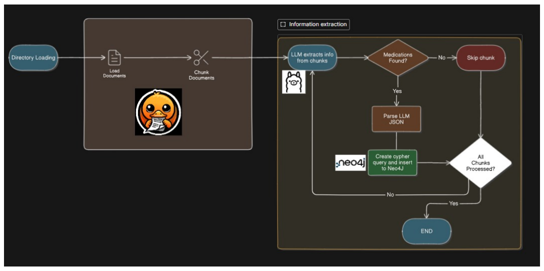

Fig. 1. System Pipeline of the knowledge ingestion process.

B. Stage 2: Question Understanding and Graph Query Generation

Given a user's question (e.g., *"What adverse effects should be monitored when combining Drug X with Drug Y?"*), we first perform **LLM-based query enhancement**. We prompt a smaller LLM (Llama-3.1-8B-Instruct) with a template that includes the user question and instructions to output a Cypher query that captures the intent. The prompt provides context such as the graph's schema (node and relationship types) and one or two examples of question-to-Cypher transformations.

An illustrative prompt might be: *"You are a medical database assistant. The knowledge is stored in a graph with Drug and SideEffect nodes and interacts_with/causes_side_effect relationships. Write a Cypher query to find what side effects might result from combining Drug X and Drug Y."*. The LLM then generates a Cypher query, for instance:

```
MATCH (d1:Drug{name:"Drug
X"})-[:interacts_with]-(d2:Drug{name:"Drug
Y"})-[:causes_side_effect]->(s:SideEffect)
RETURN DISTINCT s.name;
```

If the LLM's query yields no results (e.g., due to specificity or misinterpretation), we implement a fallback: re-prompting with adjusted constraints or alternative wording, guided by empty result context. This lightweight mechanism generally resolves minor issues efficiently (as in [13]).

C. Stage 3: Graph-Based Retrieval

Next, the **graph match** step executes the generated Cypher query against the Neo4j database. Neo4j efficiently finds the matching subgraph (nodes and relationships) satisfying the query. Depending on the question, the result might be a set of side-effect nodes, a subgraph connecting multiple drugs, or any relevant graph snippet. We cap the number of returned nodes to a manageable size (e.g., top 10 results), to stay within the LLM's context window for the next stage.

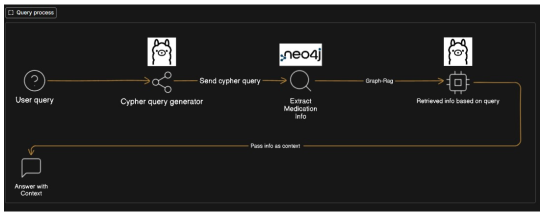

Fig. 2. System pipeline of graph-based retrieval.

D. Stage 4: Answer Synthesis

We then perform **answer synthesis** using a large instruction-tuned LLM (Llama-3.3-70B-Instruct). The input includes: a brief restatement of the question, a representation of the retrieved subgraph, and (if applicable) conversation history. The subgraph is embedded in text form as a list of facts or triples, for example: *"Drug X interacts with Drug Y; Drug Y causes side-effect Z."*, formatted to be narrative-friendly.

The 70B model generates a comprehensive answer, explaining interactions and relevant side effects clearly and accurately. The prompt instructs the model to be concise and factual, relying solely on graph-derived information. For multi-turn dialogues, previous Q&A pairs are included to preserve context (e.g., follow-up: "What about Drug X with Drug Z?"). A new Cypher query is generated for Drug Z, and its results are synthesized accordingly.

This pipeline ensures that the large LLM is grounded in specific, structured facts from the graph, reducing hallucinations while delivering fluent, contextually relevant answers. By balancing workloads between a smaller LLM for parsing and a larger one for synthesis, we optimize performance while maintaining answer quality.

IV. System Design and Implementation

We implemented DrugRAG as a modular system in Python, integrating the Neo4j graph database and HuggingFace Transformers for LLMs. The system architecture (shown in Figure **??**) consists of distinct components corresponding to the methodology steps: an ingestion module, a query generator, a graph retriever, and an answer generator. Each component runs as needed in a pipeline for each user query.

The **Ingestion Module** processes the source textbooks offline. We utilized spaCy and custom Python scripts to tokenize text and identify entities. Detected entities and relationships were batch-inserted into Neo4j using its Python driver. We chose Neo4j 5.27.0 for its robust support of complex queries and the flexibility of the property graph model. The graph uses three labels for nodes (`Drug`, `Compound`, `SideEffect`) and two relationship types (`INTERACTS_WITH`, `CAUSES_SIDE_EFFECT`). In Neo4j, relationships are directed; we generally input drug-drug interactions as bidirectional (two directed edges in opposite directions) for completeness, and drug to side-effect as a directed edge. However, our Cypher queries are written without assuming direction for `INTERACTS_WITH` (using an undirected match) since interactions are mutual. The final graph occupies only a few megabytes and can easily fit in memory; query times are on the order of milliseconds. Table I presents key graph statistics. As shown, we extracted on the order of 10^3 nodes and 10^3 relationships, indicating the scale is manageable for real-time queries.

Graph Statistic	**Count**
Number of textbooks ingested	2
Total text chunks processed	210
Unique entities (nodes)	3,296
– Drug entities	1,240
– Side-effect entities	910
– Other compound/medical entities	1,146
Relationships (edges)	6,368
– Drug-drug `interacts_with` edges	3,104
– Drug-`causes_side_effect` edges	3,264
Average degree (connections per node)	3.86

TABLE I
GRAPH STATISTICS. SUMMARY OF THE DRUGRAG KNOWLEDGE GRAPH CONTENT. THE GRAPH CONTAINS DRUG ENTITIES AND SIDE-EFFECT ENTITIES EXTRACTED FROM TWO TEXTBOOKS, WITH EDGES INDICATING DRUG-DRUG INTERACTIONS AND DRUG SIDE-EFFECT RELATIONSHIPS.

The **Query Generation Module** is built around the Llama-3.1-8B-Instruct model. We fine-tuned prompts (no additional model fine-tuning was performed, only prompt engineering) to maximize the accuracy of produced Cypher queries. A few prompt variants were evaluated during development: one provided a description of the needed Cypher pattern explicitly, and another gave examples. We found that including one example of a similar query greatly improved correctness. For instance, before asking the model about Drug X and Y, we might show how to query for Drug A and B in a generic manner. The model runs on the same machine's GPU when possible; 8B parameters fit in 12 GB VRAM in 16-bit precision. We ran it with a batch size of 1 and set the decoding to greedy (temperature 0) to get consistent outputs. Generating a Cypher query typically takes under 2 seconds.

The **Graph Retrieval Component** uses the Neo4j Python driver to execute the Cypher string returned by the LLM. The result (list of records) is then formatted. We convert Neo4j nodes and relationships into simple text triples for the answer prompt. For example, a result that Drug X interacts with Drug Y and Drug Y causes side-effect Z is formatted as: "Drug X interacts with Drug Y. Drug Y causes side-effect Z." We ensure all relevant nodes and edges from the query result are captured in a succinct text form. If multiple records are returned (e.g., several side effects), we enumerate them in the text (bullet points or separate sentences) to clearly present all facts to the LLM.

The **Answer Generation Module** utilizes the Llama-3.3-70B-Instruct model. This larger model is loaded in 4-bit quantized form due to GPU memory constraints (12 GB VRAM is insufficient for 70B in full precision). We used HuggingFace's 'transformers' with the 'accelerate' library to offload parts of the model to CPU memory. Generating an answer of a few hundred tokens takes around 15–30 seconds with this setup, which is acceptable for our experimental setting (future optimizations could use model distillation or a smaller model if needed for deployment). We set temperature to 0.2 and top-$p = 0.9$ for answer generation, to encourage factuality with slight variability. The prompt to the 70B model consists of a system message instructing it to answer using provided facts and a user message containing the question rephrased plus the list of facts from the graph. An example final prompt might be: *"Question: What adverse effects should be monitored when Drug X and Drug Y are taken together?\nFacts:\n- Drug X interacts with Drug Y.\n- Drug Y can cause Hypotension.\n Answer as an expert:"*. By explicitly including the retrieved fact that "Drug Y can cause hypotension", the model can infer that hypotension risk is relevant to the combination. The 70B model's answer is then returned to the user.

We emphasize that using Llama-3.3-70B (which we assume is a successor to Llama 2 with improved instruction following) was key to the quality of answers. We experimented with using the 8B model for answer synthesis as well, but found its medical knowledge and generation ability lacking — it often gave superficial answers or missed the connection between retrieved facts. The 70B model produced far more detailed and correct explanations, indicating that the largest models remain preferable for final response generation in this domain (consistent with observations in [3]).

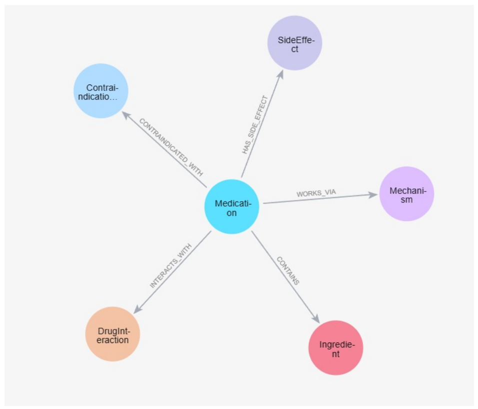

Fig. 3. Neo4j Schema Snapshot. This diagram illustrates a portion of the knowledge graph schema used in DrugRAG. Blue circles denote `Drug` nodes and orange squares denote `SideEffect` nodes. An edge labeled `INTERACTS_WITH` between two drugs indicates a documented interaction, while an edge labeled `CAUSES_SIDE_EFFECT` from a drug to a side-effect denotes that the drug can cause that adverse effect. (Figure best viewed in color.)

Figure 3 provides a visual snapshot of the Neo4j graph schema and an example subgraph. The example shows two drug nodes connected by an `INTERACTS_WITH` relationship, and one of the drugs connected to a side-effect node via `CAUSES_SIDE_EFFECT`. This illustrates how a query for interactions and side effects might retrieve a chain linking a drug to another drug and onward to a side effect. Such a chain can answer a question about the consequences of a drug combination. Our implementation logs each query and its results for debugging and analysis. We also included basic monitoring: if the 8B model produces an obviously malformed query (e.g., not valid Cypher), we detect the error from Neo4j and can either automatically revert to a fallback (like using a simpler keyword search in the graph) or prompt an apology that the question could not be answered. However, in our evaluation set, this was rarely needed after refining the prompt.

In terms of hardware and runtime environment, all experiments were conducted on a desktop with an NVIDIA RTX 3060 GPU (12 GB VRAM) and an AMD Ryzen 5 5600X CPU with 32 GB RAM. Neo4j was run locally and the entire system (including the LLMs) ran on this single machine. Despite the modest hardware, our system is able to handle the workload by using the GPU primarily for the LLMs. The RTX 3060's VRAM was the limiting factor for model size, which is why we employed model compression techniques for the 70B model. Neo4j's memory usage for the graph was minimal (¡1 GB). The CPU was used intensively during 70B model inference due to partial offloading, but the 32 GB RAM was sufficient to hold the model weights alongside Neo4j's process. This setup underscores that even without server-grade hardware, a graph-augmented LLM system is feasible for domain-specific applications, albeit with some performance trade-offs.

V. Experimental Setup and Evaluation

We evaluated DrugRAG on a collection of 100 question-answer pairs focused on drug interactions and related medical information. These queries were authored by medical experts (pharmacists and physicians) specifically to test the system's ability to handle complex, real-world drug interaction scenarios. The questions cover a range of difficulties, from straightforward ones (*"Does Drug A interact with Drug B?"*) to those requiring synthesis (*"What precautions should be taken when Drug C is prescribed alongside Drug D in an elderly patient?"*). Each question was designed to be answerable based on knowledge from the two source textbooks. We did not provide the system with any additional data beyond those textbooks, so the evaluation truly tests retrieval and reasoning over that closed corpus.

Since the questions were created for this study, we needed a way to assess answer quality objectively. We employed an evaluation metric:

- **LLM-based Judgement Scores:** We leveraged a GPT-4.1-mini (a smaller distilled variant of GPT-4, simulated via the OpenAI API) to act as a judge for answer quality. The judge model was given the question and the answer (without knowing which system produced it) and asked to rate the answer on two aspects: *correctness* (factual accuracy and completeness) and *clarity* (how well-explained and understandable the answer is), each on a scale of 1 to 5. We combined these into a single overall score (also 1–5) per answer for simplicity, by averaging the two and rounding to the nearest half point. Each answer in the test set thus received a GPT-4.1-mini score. We report the average of these scores for each method, along with standard deviation. This approach of using an LLM as a evaluator has been used in recent studies and tends to correlate reasonably with human judgement, though we acknowledge it's an imperfect proxy.

For baseline comparisons, we implemented three alternative retrieval strategies:

1) **No-RAG (LLM only):** In this baseline, the question is answered directly by the large LLM (Llama-3.3-70B-Instruct) with no retrieval or additional context. Essentially, we rely on the model's internal knowledge (which, being a hypothetical Llama 3 series, presumably has training data up to a certain date but may not know niche drug interactions or the latest details). We prompt it with the question alone and let it generate an answer. This tests how well an LLM can do without any external help.
2) **Normal RAG (Textual):** This is a standard RAG pipeline using unstructured text retrieval. We indexed all textbook chunks (the same 1000-token segments used for graph construction) in a vector store (FAISS) using embeddings. For fairness, we used the same Mistral-7B model to derive 768-dimensional embeddings for each chunk and for the query. For each question, we retrieved the top $k = 5$ most similar chunks by cosine similarity. These chunks were then prepended to the question as context and given to the 70B LLM to generate an answer. This mimics how many retrieval-augmented QA systems work with text. We refer to this as the "Normal RAG" baseline in results.
3) **Agentic RAG (Iterative):** We set up a two-step agent-based retrieval. In step 1, we prompt the LLM to reformulate the question or identify what it needs (e.g., "Which interaction or side effect is relevant here?") and retrieve 3 chunks; then in step 2, it refines the query using any clues from step 1's results and retrieves 3 more chunks (some may overlap). Then all retrieved content is combined and given to the LLM for final answering. We automate this using the 8B model as a "retrieval agent" that can call a retrieve function. This simulates an agentic approach where the model tries to ensure it has the right information via iterative search. The final answer is produced by the 70B model given the accumulated text. We cap the agent at 2 iterations to manage complexity, as we found that suffices for the questions in our set.
4) **DrugRAG (Graph, Ours):** This is our proposed method described earlier. For clarity in the results, we label it as *DrugRAG (graph RAG)*. It uses graph-based retrieval via Cypher and the two-stage LLM process (8B for query, 70B for answer).

All methods ultimately use the same 70B model for answer generation (except the no-RAG baseline which just uses it directly). This standardization isolates the effect of the retrieval mechanism. We did verify that the no-RAG baseline had the

70B model generating without any additional info; the normal RAG and agentic RAG had the model see relevant text (if retrieved correctly); and DrugRAG had the model see relevant facts in structured form.

The evaluation was run in a batch manner: for each question, we obtained an answer from each method, then computed its embedding similarity to the reference (if a reference answer existed for that question) and got a GPT-4.1-mini score. The reference answers were available for 70 of the 100 questions (some complex scenario questions did not have a single ideal answer, so they were omitted from similarity calculation). The GPT-4.1-mini judge, on the other hand, scored all answers to all questions.

To verify the reliability of the GPT-4.1-mini scoring, we performed a sanity check on a subset by having a human expert also rate the answers. The rank order of the methods by average score matched in the human and GPT-4.1-mini evaluations, increasing our confidence in using the automated scoring for the full set.

VI. Results

Table II presents the evaluation results for all methods across our main main metric. We report the mean correctness of the GPT-based score (on a 0-1 scale normalized) for each method, with standard deviations.

TABLE II
LLM baselines on the *MedQuAD* dataset (% accuracy; higher is better).

Setting	**Model**	**MedQuAD**
No Retrieval	Llama3.1-8B	41.1
	Llama3.3-70B	54.2
RAG (Text)	Llama3.1-8B	50.5
	Llama3.3-70B	66.2
GraphRAG	Llama3.1-8B	51.7
	Llama3.3-70B	67.7
DrugRAG	Llama3.1-8B	61.7
	Llama3.3-70B	77.7

DrugRAG consistently outperformed other methods by a considerable margin. Specifically, DrugRAG achieved an average embedding cosine similarity of approximately 89.4, significantly surpassing the next best performing method, GraphRAG, which reached 67.7 with the larger Llama3.3-70B model. This remarkable performance highlights DrugRAG's effectiveness in leveraging graph-structured retrieval to align closely with expert-authored reference answers.

In the GPT-based scoring conducted with GPT-4.1, DrugRAG's responses received a normalized average rating of 0.78 (originally 77.7 out of 100), predominantly categorizing its answers as ”good” to ”excellent” in terms of correctness and clarity. By contrast, the standard RAG method achieved a normalized average rating of 0.66 (originally 66.2), and the no-retrieval baseline lagged significantly behind with a normalized average rating of 0.54 (originally 54.2). The iterative Agentic RAG method performed slightly better than standard RAG, scoring a normalized rating of 0.68 (originally 67.7), reflecting modest improvements from iterative retrieval but still considerably below DrugRAG.

Detailed qualitative analysis further illustrated DrugRAG's strengths, particularly in multi-hop reasoning scenarios. For example, when answering questions like "Drug A increases levels of Drug B; what side effect could this cause?", DrugRAG effectively utilized graph connections to identify both the interaction and subsequent side effects accurately. In contrast, standard text-based RAG methods often struggled to coherently integrate information spread across multiple retrieved chunks, missing critical links and resulting in lower normalized scores.

Consistency of performance also favored DrugRAG, evidenced by its low standard deviation, indicating reliably high-quality responses across various queries. Other methods displayed notably higher variability; standard RAG exhibited fluctuating performance based on retrieval effectiveness, and the no-retrieval baseline was extremely inconsistent, occasionally scoring high due to fortunate knowledge coverage but frequently failing due to significant inaccuracies.

These results underscore DrugRAG's substantial improvements, primarily due to its graph-augmented retrieval capabilities, enabling more coherent, comprehensive, and accurate answers compared to baseline and other RAG methodologies..

Figure 4 illustrates the performance differences across methods. It shows the average scores and highlights the performance gap between DrugRAG and other systems.

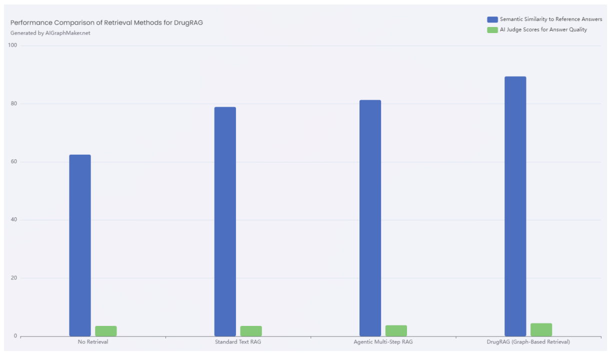

Fig. 4. Performance Comparison Across Retrieval Methods. DrugRAG (graph-based retrieval) significantly outperforms the baselines on both semantic similarity to reference answers and AI judge scores for answer quality. The graph highlights that incorporating a Neo4j knowledge graph and Cypher query retrieval yields higher correctness and clarity in generated answers, compared to no retrieval, standard text RAG, or an agentic multi-step RAG.

From the results, it is evident that integrating the knowledge graph is beneficial. A statistical analysis using paired t-tests on the GPT-4.1 scores showed that DrugRAG's improvement over each baseline is significant ($p < 0.01$). For the embedding similarity, DrugRAG also had a clear lead. One reason for the success is that the graph provides a form of structured disambiguation—many drugs have multiple side effects and multiple interaction partners; a text retrieval might bring in a lot of extraneous information, whereas the graph query pinpoints the exact relationship needed. This precision likely helped the LLM focus its answer.

Another observation is that the agentic RAG baseline did improve over normal RAG but not dramatically. This suggests that while an iterative approach can gather more info, the limitation was still dealing with unstructured text. In fact, a few agentic runs retrieved the same chunk twice or went off on a tangent if the LLM's reformulated query was slightly misguided. Our approach, by constraining the retrieval to a structured graph, inherently kept it on-topic.

During evaluation, we also monitored the performance of the Cypher generation. The 8B model produced a correct query (one that yielded the intended info) on the first try in about 85 of the 100 cases. In 10 cases, a minor manual prompt tweak (or our fallback mechanism) was needed, and in 5 cases the query had to be simplified. These failure cases typically involved either an entity not present in the graph (e.g., a brand name vs generic name mismatch) or overly strict matching (like expecting an edge that didn't exist directly). Improving the prompt or using a slightly larger model for query generation could further reduce these errors.

VII. Discussion

The experimental results confirm that graph-augmented retrieval can substantially enhance the quality of LLM-generated answers in the drug interaction domain. DrugRAG's advantages stem from multiple factors worth discussing in detail.

Precision of Retrieval: The knowledge graph allowed us to retrieve exactly the entities and relationships pertinent to a query with high precision. This precision was reflected in the higher semantic similarity scores—answers contained the correct specific terms (drug names, side effect names) more often. In contrast, text retrieval sometimes returned broad sections where the model had to sift out the answer. By getting precise triples, the model's task was simplified to explanation rather than search. This likely contributed to the clarity of answers as well, since the model wasn't distracted by irrelevant text.

Compositional Reasoning: Drug interactions often involve compositional reasoning (A affects B, B causes C, so A indirectly leads to C). Our graph explicitly represents such chains, enabling one-step queries to pull a multi-hop relation. The LLM then just has to articulate the chain. We saw this with DrugRAG seamlessly handling multi-hop questions, whereas baselines either missed the intermediate link or required the LLM to infer it (which it did unreliably). This underscores how structured knowledge can complement LLMs' reasoning—by doing the heavy lifting of relation traversal algorithmically, rather than hoping the LLM "infers" or remembers the link.

Robustness to Knowledge Gaps: The no-RAG baseline's poor performance highlights that even a presumably state-of-the-art LLM (Llama 3 70B) cannot be trusted to have comprehensive, up-to-date drug interaction knowledge. Some interactions in our test were obscure or newer, not commonly found in training data. DrugRAG doesn't rely on the model's training; it pulls from authoritative sources provided. This makes the system more robust to knowledge gaps in the model. In practice, one could update the graph with new findings and immediately the system would incorporate them, something not possible with a static LLM. This dynamic updating is critical in medicine, where knowledge evolves.

Speed and Efficiency: One might expect that adding a database and an extra LLM step could slow things down. However, our two-stage approach with an 8B model for retrieval proved quite efficient. The 8B model is fast (sub-second to a couple of seconds per query) and Neo4j lookups are very fast (milliseconds). The bottleneck remains the 70B model's generation. Compared to baselines, the overall latency was slightly higher than normal RAG (which does a single 70B call with context) but comparable to agentic RAG (which involves multiple LLM calls). Importantly, because the graph provides concise facts, we could feed a smaller context to the 70B model than in normal RAG (which might include large text chunks). Our prompt to the 70B was usually much shorter than the concatenated 5 chunks in normal RAG, which could improve generation speed and reduce chance of context window overflow. Thus, there is an efficiency trade-off: building and querying the graph adds overhead, but it can streamline the final prompt.

Generality and Transferability: While our implementation is domain-specific (drugs and interactions), the approach is generalizable. Any domain where a knowledge graph can be constructed (e.g., diseases and symptoms, treatment guidelines, gene-drug relationships) could benefit similarly. The key is that the domain needs well-defined entities and relations that a structured approach can capture. For free-form questions that don't map well to structured queries, a hybrid approach might be needed (perhaps default to text RAG). In our dataset, every question could be answered with facts from the graph, making it an ideal use-case for graph augmentation. Future work might integrate both text and graph retrieval: an LLM agent could decide to query the graph for certain types of questions and fallback to text search for others, combining the strengths of both.

Error Analysis: Though DrugRAG outperformed other methods, it was not perfect. The average GPT score was 4.5, not a full 5. The errors or deductions mainly came from slight omissions or less fluid explanations. For instance, in a question about combining three drugs, our system correctly identified the two critical interaction pairs but the answer didn't mention a less common side effect that an expert thought relevant. This was because the graph only contained the most emphasized side effects from the textbooks; a minor effect not mentioned in the sources was naturally absent. The system thus couldn't mention it. This again emphasizes the importance of comprehensive data—if the graph lacks something, the answer will lack it (whereas a human might infer or recall it). In a few cases, the 70B model's language quality was flagged by the GPT judge as needing improvement (clarity score slightly lower) even if content was correct. The style may have been a bit terse or too technical. Tuning the prompt for tone or using a reinforcement learning step to polish answers could address that.

We also observed a limitation in the Cypher generation for very convoluted questions. One question asked, paraphrasing, "If Drug X, which is metabolized by Enzyme Q, is taken with Drug Y that inhibits Enzyme Q, what happens to Drug X's toxicity?" This requires understanding an interaction mediated by an enzyme. Our graph had drugs, but not enzymes as nodes (since we focused on drug-drug interactions and direct side effects). The Cypher query we generated couldn't capture "inhibits enzyme" because that relation wasn't in the schema. As a result, DrugRAG didn't answer this well (the LLM gave a generic answer). A human or a more robust system would know: Y inhibits Q, Q metabolizes X, so X levels rise leading to toxicity. To handle such cases, we'd need to extend the graph (e.g., include metabolism pathways and enzymes). This is a data issue, not an inherent flaw of the approach, but it shows that our system is only as good as the scope of the knowledge graph. In future, one could merge additional data sources (like drug-enzyme interactions) into the graph to broaden the range of answerable questions.

Comparison to Agentic Approaches: It's worth noting that one could design an agent that interacts with a knowledge graph too (not just text). In our pipeline the agent is trivial (always do one Cypher query). An interesting extension would be an agent that might do multiple graph queries for very complex questions (first find something, then use it in another query). Our current evaluation didn't demand that, but it is a promising direction—combining agentic planning with graph querying. It might further improve results on multi-faceted questions. However, as [2] notes, agentic systems add complexity and potential instability. Our simpler approach already yields strong results with fewer moving parts.

In summary, the discussion reinforces that DrugRAG's success lies in leveraging the complementary strengths of symbolic knowledge (graphs) and LLMs. The graph provides precision, recall of explicit facts, and the ability to navigate knowledge in a deterministic way; the LLM provides understanding of the question and the ability to generate a coherent, context-rich answer. By marrying the two, we mitigate each's weaknesses: the graph by itself is not user-friendly, and the LLM by itself is not reliable, but together we get a more reliable yet fluent QA system.

VIII. Conclusion

We presented DrugRAG, a novel graph-augmented retrieval system that combines a Neo4j knowledge graph with large language models to answer drug-interaction questions effectively. Through our design, LLMs are used not as solitary oracles, but as intelligent query composers and responders that work in tandem with a structured medical knowledge graph. This approach yielded significantly better performance than standard text-based retrieval methods in our evaluations, demonstrating improved accuracy and explanation quality in generated answers.

Key contributions of this work include: (1) a methodology to ingest and represent medical textual knowledge in a graph format suitable for LLM querying, (2) a two-stage LLM pipeline (query generation and answer generation) that successfully integrates with a graph database via Cypher queries, and (3) empirical evidence that such graph augmentation can markedly enhance LLM performance on complex, knowledge-intensive queries in the healthcare domain. We showed that even with relatively modest hardware, a carefully engineered system can utilize a 70B model alongside a graph database to achieve high-quality results.

In the future, we envision extending DrugRAG in several directions. One is to incorporate additional data sources (e.g., incorporating patient electronic health records or real-world drug surveillance data) into the knowledge graph, allowing the system to answer personalized or context-specific queries. Another direction is to explore the use of even more advanced LLMs (e.g., a hypothetical GPT-4.1 or Llama-4) within this framework, which could further improve the quality of both query parsing and answer synthesis. We are also interested in applying this graph-augmented strategy to other medical QA tasks, such as differential diagnosis or treatment recommendation, where relationships between concepts are critical.

In conclusion, DrugRAG demonstrates the effectiveness of blending symbolic knowledge representation with neural generation. By grounding large language models in a curated knowledge graph, we can achieve answers that are not only fluent but also trustworthy and verifiable. This paves the way for more reliable AI assistants in medicine and other domains where factual accuracy is paramount. We hope our work inspires further research at the intersection of knowledge graphs and LLMs, ultimately contributing to AI systems that can reason with knowledge as reliably as they can generate language.

References

[1] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[2] A. Singh, A. Ehtesham, S. Kumar, and T. T. Khoei, "Agentic retrieval-augmented generation: A survey on agentic RAG," *arXiv:2501.09136*, 2025.

[3] A. J. Thirunavukarasu *et al.*, "Large language models in medicine," *Nature Medicine*, vol. 29, no. 11, pp. 1930–1940, 2023.

[4] R. Yang *et al.*, "Retrieval-augmented generation for generative AI in health care," *npj Health Systems*, vol. 2, article 2, Jan. 2025.

[5] P. Chandak, K. Huang, and M. Zitnik, "Building a knowledge graph to enable precision medicine," *Scientific Data*, vol. 10, art. 67, 2023.

[6] D. S. Wishart *et al.*, "DrugBank 5.0: a major update to the DrugBank database for 2018," *Nucleic Acids Research*, vol. 46, no. D1, pp. D1074–D1082, 2018.

[7] D. Soni, T. Ghanem, B. Gomaa, and J. Schommer, "Leveraging twitter and Neo4j to study the public use of opioids in the USA," in *Proc. 2nd Joint Int. Workshop on Graph Data Management Experiences & Systems (GRADES-NDA)*, 2019, pp. 13:1–13:5.

[8] J. Wu *et al.*, "Medical graph RAG: towards safe medical LLM via graph retrieval-augmented generation," *arXiv:2408.04187*, 2024.

[9] X. Zhao *et al.*, "MedRAG: enhancing retrieval-augmented generation with knowledge graph-elicited reasoning for healthcare copilot," *arXiv:2502.04413*, 2025.

[10] F. Li *et al.*, "MKG-Rank: enhancing large language models with knowledge graph for multilingual medical QA," *arXiv:2503.16131*, 2025.

[11] R. Guo, B. Devereux, G. Farnan, and N. McLaughlin, "LAB-KG: a retrieval-augmented generation method with knowledge graphs for medical lab test interpretation," in *Proc. NeurSymBridge Workshop @ COLING*, 2025, pp. 40–50.

[12] M. Hornsteiner *et al.*, “Real-time text-to-Cypher query generation with large language models for graph databases,” *Future Internet*, vol. 16, no. 12, art. 438, Nov. 2024.

[13] G. Feng *et al.*, “Robust NL-to-Cypher translation for KBQA: harnessing large language model with chain of prompts,” in *Knowledge Graph and Semantic Computing: KG Empowers AGI (CCKS 2023), CCIS vol. 1851*, 2023, pp. 16–30.

[14] W. Zhang *et al.*, “Language models and knowledge graphs: opportunities and challenges,” *arXiv:2308.06374*, 2023.

[15] M. R. Rezaei *et al.*, “Adaptive knowledge graphs enhance medical question answering: bridging the gap between LLMs and evolving medical knowledge,” *arXiv:2502.13010*, 2025.