PharmaGraph: Drug Interaction Analysis Using Knowledge Graphs and Restricted LLMs

Youssef Hassan Abdelmaksoud, Ziad Moataz Samy, Ahmed Kamal Shaaban
Computer Science and Engineering Department, Nile University, Cairo, Egypt
Email: {y.hassan2156, z.moataz2156, a.kamal2102}@nu.edu.eg

***Abstract*—Drug-drug interactions (DDIs) pose significant risks to patient safety, especially for those on complex medication regimens. We present *PharmaGraph*, a system that leverages a knowledge graph of drug interactions and retrieval-augmented language modeling to assist in DDI analysis. PharmaGraph integrates a curated drug knowledge graph (built from DrugBank data) with biomedical named-entity recognition and various search strategies to answer user queries about DDIs. The system provides four main functionalities: (1) checking if a specified pair of drugs interacts, (2) finding all interaction partners for a given drug, (3) checking a medication against a list of other drugs for potential interactions, and (4) recommending a safe alternative drug for a given condition considering a patient's current medications. We employ a BioNLP-trained entity recognizer (SpaCy BC5CDR) to extract drug and condition entities from user input, and we explore multiple knowledge retrieval methods—from direct graph queries to advanced Retrieval-Augmented Generation (RAG) techniques incorporating graph context. In experiments on a benchmark of known drug interactions, PharmaGraph achieves high accuracy (up to 97% F1-score) in identifying interactions. It offers clear interaction explanations and safe medication recommendations, demonstrating the effectiveness of combining knowledge graphs with large language models for clinical decision support in medication management.**

***Index Terms*—Drug-Drug Interaction, Knowledge Graph, Retrieval-Augmented Generation, Biomedical NLP, Clinical Decision Support**

I. Introduction

Patients receiving multiple medications are at risk of adverse effects due to drug-drug interactions (DDIs). Identifying potential DDIs is a critical task for clinicians and pharmacists to prevent harmful outcomes. Large knowledge bases such as DrugBank compile extensive information on known DDIs, totaling thousands of documented interacting drug pairs [1, pp. D1265–D1275]. However, utilizing this knowledge effectively in clinical practice can be challenging. Clinicians often rely on electronic references or interaction checker tools, which may list interactions without context or patient-specific guidance.

Recently, large language models (LLMs) have shown promise as tools for medication guidance, particularly in explaining drug interactions and advising on therapy adjustments [2]. LLMs can produce human-like explanations, but their responses risk factual errors if not *grounded* in a reliable knowledge source. In the context of DDIs, an LLM might incorrectly assert an interaction or overlook a critical one, especially for less common drug pairs, unless it is augmented with up-to-date interaction data.

In this work, we propose **PharmaGraph**, a system that combines a structured drug interaction knowledge graph with LLM-based generation to support safe medication decisions. PharmaGraph addresses two main needs: (1) accurately detecting and explaining whether a given combination of drugs can interact, and (2) recommending alternative medications that treat the same condition but avoid dangerous interactions. Unlike a basic lookup tool, our system integrates multiple AI components to understand user queries (including free-text descriptions of a patient's case), retrieve relevant interaction knowledge, and generate contextualized advice. By building on DrugBank's authoritative interaction data [1] and state-of-the-art biomedical NLP techniques, PharmaGraph provides a more intelligent DDI analysis platform.

We implemented four user-facing functionalities in PharmaGraph: checking interaction between two drugs, finding all interactions of a single drug, checking one drug against a list of others, and recommending a safe drug for a given condition. The backend uses a knowledge graph derived from DrugBank to store drug entities and their known interactions. User queries are processed through a biomedical named entity recognition (NER) model to extract drug names and medical conditions. Then, various retrieval strategies are applied to obtain relevant interaction information. We explore direct graph queries as well as Retrieval-Augmented Generation (RAG) approaches that leverage the knowledge graph for improved reasoning. Finally, for recommendation queries, an LLM is used to propose a suitable medication while ensuring it does not interact with the patient's current drugs by cross-referencing the knowledge graph.

We evaluate the system on a benchmark dataset of known DDIs and demonstrate that PharmaGraph achieves high precision and recall in interaction detection (over 95% F1). The integration of the LLM enables it to produce natural language explanations and recommendations that are consistent with the knowledge graph. Our results highlight that combining knowledge graphs with generative AI under strict grounding constraints can yield an effective clinical decision support tool for medication management. In the following sections, we discuss related work, detail the system's methodology and design, present evaluation results, and outline potential improvements.

II. Related Work

A. Drug-Drug Interaction Resources and Prediction

Comprehensive resources like DrugBank and similar databases provide curated collections of known drug interactions, which are widely used in research and clinical practice [1]. Beyond static databases, a large body of research has focused on DDI prediction using computational methods. For example, Zhong *et al.* [3] developed a multi-relational knowledge graph embedding method with contrastive learning to predict potential DDIs. Similarly, recent graph neural network models (e.g., KnowDDI) incorporate biomedical knowledge graphs to improve prediction performance and interpretability of DDIs by identifying important subgraph connections between drug pairs [4]. These works aim to discover new or previously unrecognized interactions by learning patterns in known interaction networks. In contrast to such predictive models, our work focuses on harnessing a knowledge graph of *known* DDIs to answer user queries and provide decision support. Rather than predicting novel interactions, PharmaGraph ensures high accuracy on identifying documented interactions and emphasizes user-facing explanation and advice.

B. Retrieval-Augmented Generation and Knowledge Graphs

Integrating external knowledge with LLMs through retrieval-augmented generation (RAG) has emerged as a promising technique for knowledge-intensive tasks. Lewis *et al.* introduced the RAG framework for open-domain question answering [5], where relevant text passages from a large corpus are retrieved and fed into a generative model to improve factual accuracy. While standard RAG uses unstructured text and dense vector retrieval, recent extensions have explored structured knowledge integration. Microsoft Research's *GraphRAG* approach incorporates knowledge graph information to enhance retrieval and multi-hop reasoning for question answering [6]. By building a graph-based index of entities and relationships from documents, GraphRAG enables an LLM to navigate complex queries that require connecting multiple facts. Another advance is *LightRAG*, proposed by Guo *et al.* [7], which incorporates graph structures into the retrieval process to achieve more efficient and contextually relevant generation. LightRAG uses a dual-level retrieval strategy: it captures low-level details about specific entities and high-level knowledge about their relationships, thus improving both speed and answer coherence.

Our system draws inspiration from these approaches by using a domain-specific knowledge graph of DDIs to inform the LLM. We implement several retrieval strategies: from a basic direct lookup of graph edges to more advanced methods that provide the LLM with graph-based context. We evaluate variants such as a GraphRAG-style retrieval (leveraging neighborhood subgraphs of the drugs in question) and a query-refinement RAG (denoted as *RQ-RAG*) where the LLM first helps reformulate the query or focus, as well as a hybrid *LightGraphRAG* which adapts LightRAG principles to our DDI graph domain. To our knowledge, these methods have not been previously applied to the specific problem of interactive drug interaction queries.

C. Biomedical NLP for Entity Recognition

Understanding user queries about medications and conditions requires accurate identification of biomedical entities (drug names, diseases, etc.). There has been extensive work on biomedical named entity recognition (NER). SciSpaCy is a library that provides robust NLP models for biomedical text, including NER models trained on scientific and clinical datasets [8]. In particular, SciSpaCy offers a pre-trained model for recognizing chemical and disease entities based on the BioCreative V CDR (Chemical-Disease Relation) corpus [9]. The BC5CDR dataset contains annotations for chemicals (including drug names) and diseases in biomedical literature, and NER models trained on it have achieved strong performance on identifying these entity types.

In PharmaGraph, we utilize a spaCy model (en_ner_bc5cdr) from SciSpaCy to extract drug names and medical conditions from free-form text input. This allows the system to handle natural language queries (e.g., "I'm taking *Drug A* and *Drug B* together, is it safe?") by converting them into structured data (identifying "Drug A" and "Drug B" as the drug entities of interest). High recall in entity extraction is crucial: if a drug name is missed or mis-recognized, the subsequent knowledge graph query might fail. We therefore favor a domain-specific NER approach over general-purpose NER to ensure that even uncommon drug names or medical terms are recognized.

III. Methodology

A. Knowledge Graph Construction

The core of PharmaGraph is a knowledge graph (KG) that encodes known drug-drug interactions. We constructed this KG from the DrugBank database [1], focusing on approved small-molecule drugs and their pairwise interactions. Each drug is represented as a node in the graph, identified by its generic name (with DrugBank identifiers as secondary identifiers). An undirected edge between two drug nodes indicates a documented interaction between those drugs. We treat interactions as a symmetric relationship INTERACTS_WITH (since if drug A interacts with B, typically B interacts with A). In addition to the graph structure, we store interaction detail attributes on each edge, such as a brief description of the interaction (e.g., the effect or mechanism) and severity level if available. These details are extracted from DrugBank's interaction annotations. The resulting graph provides a comprehensive knowledge base of known DDIs that can be queried programmatically.

B. Interaction Query Functions and Retrieval Strategies

PharmaGraph provides four key functions for DDI analysis, all of which rely on querying the DrugBank-derived knowledge graph (using NetworkX in Python for graph operations). We briefly describe each functionality and the retrieval strategy used:

1) **Two-Drug Interaction Check:** Given two drug names, the system checks if an edge exists between the corresponding nodes in the graph. A direct graph lookup is performed to determine if the pair is known to interact. If an interaction is found, the stored description and severity are retrieved; if not, the system responds that no known interaction exists for that pair. This direct method is fast and precise, returning results strictly from the graph data.
2) **All Interactions of a Drug:** Given a single drug, the system retrieves all one-hop neighbors of that drug in the knowledge graph. These neighbors represent all drugs that have documented interactions with the query drug. The result is a list of interacting drugs (and optionally the interaction details for each). By traversing the graph, this method achieves full coverage of known interactions for the queried drug.
3) **Interaction Check Against a List:** For a medication to be taken alongside a list of other drugs, the system checks the main drug against each drug in the list. This is essentially a batch repetition of the two-drug check: for each drug in the list, query the graph for an interaction with the main drug. Any interactions found are reported (with details). If none of the pairwise checks yield an interaction, the main drug is deemed safe with respect to that list. This functionality again relies purely on direct knowledge graph lookups and will flag interactions if and only if they exist as edges in the graph.
4) **Safe Alternative Recommendation:** This more advanced function is described in detail in Section III-C. In brief, given a patient's current medications and a new diagnosis, the system aims to recommend a drug for the new condition that does not interact with the current drugs. It does so by retrieving candidate drugs for the condition (from a mapping of indications or the LLM's knowledge) and then filtering them against the knowledge graph to eliminate any that have interactions with the current regimen. An LLM is then used to select an appropriate drug from the safe candidates and provide a rationale.

All of the above functionalities fundamentally depend on querying the DDI knowledge graph to obtain interaction information. However, we explore multiple retrieval and reasoning strategies to enhance how results are generated, especially for explanatory outputs:

- **Direct Graph Lookup (Direct):** The simplest strategy is as described in functions 1–3: directly query the graph for the relevant edge or neighbors. This method uses no language model and yields a binary answer or list directly from the database. For example, for the query "Does Drug X interact with Drug Y?", the Direct method checks if edge (X, Y) exists in the KG and returns a yes/no along with the stored description if yes. Direct lookup is highly precise and efficient, but provides minimal context beyond the raw interaction information.
- **Neighbor Search (Neighbor):** In this approach, we consider the immediate interaction network around the query entities. For a two-drug query where a direct edge might not exist, the Neighbor strategy retrieves the sets of known interaction partners of each drug (one-hop neighbors in the graph). Identifying any common neighbors or related interaction patterns can provide context. While an indirect connection (e.g., A interacts with C and B interacts with C) does *not* imply that A and B interact, such information can be useful for the LLM to reason or to highlight pharmacological similarities (for instance, if both drugs interact with a third drug C, it might suggest caution or a common pathway). In practice, we use neighbor search to gather additional context for the LLM-based methods rather than as a standalone user answer: the system would not report an interaction unless a direct edge is present, but it may supply neighbor information to the LLM for a more nuanced explanation.
- **GraphRAG:** This method follows the Graph Retrieval-Augmented Generation paradigm [6]. For a given query, we retrieve a subgraph of the knowledge graph that is relevant to the query and feed it as context to the LLM. For example, if the user asks about two drugs A and B, we gather not only the direct edge A–B (if present) but also the interaction neighbors of A and of B (i.e., the set of drugs interacting with A, and with B). We then construct a structured summary of this subgraph as part of the LLM prompt, e.g.: "*Drug A interacts with [list of A's neighbors]; Drug B interacts with [list of B's neighbors]; (if A and B directly interact, include that detail as well).*" The LLM uses this graph-derived information to generate a comprehensive answer. Because the LLM is explicitly given the known interactions of each drug, it can mention relevant facts (such as known interaction partners or the lack of a direct interaction) and provide reasoning. This approach incurs more computational cost (due to invoking the LLM), but it yields a detailed, human-readable response and is more robust to varied question phrasing. The answer remains grounded in the retrieved facts, as the LLM is constrained to use the provided graph information.
- **RAG with Query Refinement (RQ-RAG):** In some cases, user queries may be complex, implicit, or poorly structured. We experimented with a two-step LLM approach where the model first reformulates or analyzes the user query to determine the core question and entities (acting as a query refiner), and then performs a GraphRAG retrieval based on that refined query. For example, a user might ask, "*I take Drug A for condition X and now need Drug B; is that safe?*" The first LLM pass would interpret that the essential question is about an interaction between Drug A and Drug B (identifying A and B as the drug entities of concern). After this clarification, the system retrieves relevant graph data as in the GraphRAG approach (neighbors and direct edge for A and B) and feeds both the refined query

and the graph context into a second LLM prompt to produce the final answer. This RQ-RAG strategy aims to improve accuracy in scenarios where the raw user input might confuse straightforward entity extraction or retrieval (especially if the query is long and narrative). By having the LLM focus the query first, we reduce the chance of retrieving irrelevant information or missing the key interaction being asked about.

- **LightRAG (Text-based Retrieval):** We also include a retrieval baseline inspired by LightRAG [7] that does not explicitly use the graph structure during query time. In this approach, all known interactions are encoded as textual triples or sentences (e.g., "*Drug A interacts with Drug B (causes elevated risk of arrhythmia)*"). These sentences are indexed in a vector database for semantic search. Given a query, the system performs a dense retrieval of the most relevant interaction statements based on the query embedding (similar to a standard RAG pipeline on unstructured data). The retrieved textual facts are then provided to the LLM to generate an answer. This method leverages the same data but ignores the graph topology at query time, treating the knowledge as a set of text snippets. We use this as a lightweight baseline to see how purely text-based retrieval compares to graph-aware retrieval in our domain.
- **LightGraphRAG (Hybrid):** We introduce *LightGraphRAG*, a hybrid strategy that combines graph context with the efficiency of LightRAG. Like LightRAG, we pre-index textual representations of interactions with a vector store. However, we constrain or enrich the semantic search with graph structure. For example, when a specific drug or pair is queried, we limit the search space to sentences involving those drugs or their neighbors, or we boost those results. In practice, LightGraphRAG retrieves only a few highly pertinent interaction statements from the knowledge graph (by semantic similarity to the query) rather than entire neighbor lists. These retrieved facts, carrying both the drug names and their relationship, are given to the LLM to formulate the answer. The intuition is that this "lighter" retrieval provides just the key pieces of knowledge the LLM needs, making its job easier and faster than absorbing a large subgraph context, while still ensuring the answer is grounded. We found that LightGraphRAG often preserved the accuracy of full GraphRAG while improving response speed.

Each of the above methods ultimately produces an answer that could be a simple list (for the direct methods) or a narrative explanation (for the LLM-generated methods). The system is designed to choose a retrieval strategy appropriate to the query type: for straightforward lookups (like listing all interactions of one drug), a direct graph query is sufficient and fastest. For more nuanced queries or those requiring explanation (e.g. "why are these two drugs risky to combine?"), a GraphRAG or LightGraphRAG approach is employed to provide context for the LLM. Crucially, the outputs from the knowledge graph (such as the known interaction descriptions) are always included in the prompt to keep the LLM grounded in factual data, thereby "restricting" the LLM from hallucinating information outside the knowledge graph.

C. *Safe Drug Recommendation via LLM*

The most advanced functionality of PharmaGraph is recommending a safe alternative drug given a patient's current medications and a new treatment need (diagnosis). This task is formulated as follows: the user provides a list of drugs that the patient is already taking (current regimen) and a condition that needs to be treated with an additional drug. The goal is to suggest a drug for that condition that has no dangerous interactions with any of the current drugs.

Our approach to this task combines domain knowledge with LLM reasoning in multiple steps:

1) **Candidate Drug Identification:** First, we determine a set of candidate drugs that are commonly used to treat the given condition. We compiled a simple condition-to-drug mapping from standard clinical guidelines and DrugBank indications. For example, if the condition is *hypertension*, the candidate drugs might include several first-line antihypertensives (ACE inhibitors, beta blockers, diuretics, etc.). If the condition is recognized in our mapping, we retrieve a list of possible drugs (typically on the order of 5–10 drug names) indicated for that condition. If we do not have a predefined list for the condition, we fall back to querying the LLM itself for suggestions (the LLM has general medical knowledge and can usually list common treatments for many diseases).
2) **Interaction Filtering via KG:** Once we have the candidate drugs for the condition, we filter them against the patient's current drug list using the knowledge graph. Specifically, for each candidate, we check it for interactions with each drug in the current regimen. Any candidate that has a known interaction with any of the current drugs is eliminated from consideration. For example, if the patient is on Drug X and one candidate for the condition is Drug Y which interacts with X (edge X–Y exists in the KG), then Y is removed from the pool. This step ensures that only interaction-free (with respect to known DDIs) options remain.
3) **LLM-based Recommendation:** The filtered list of safe candidates, along with context about the patient's situation, is then passed to an LLM which selects the most suitable drug and provides a rationale. The LLM prompt includes the condition, the current drugs, and the list of candidates that are considered safe, and asks the model to recommend the best option from those. We instruct the LLM to explain why the chosen drug is appropriate (mentioning that it does not interact with the current medications and is a valid treatment for the condition). If multiple options are equally good, the LLM may mention alternatives or clarify its choice.

4) **No Safe Option Handling:** In the rare scenario that every candidate drug for the condition interacts with the current regimen (i.e., the knowledge graph indicates potential interactions for all common treatments of the condition), the LLM is instructed to acknowledge this and suggest a different approach. Rather than recommending a dangerous option, it will respond that finding a completely interaction-free medication is difficult and may suggest consulting a specialist or considering non-pharmacological treatments or careful monitoring. This ensures the system does not provide unsafe advice even when the ideal solution may not exist.

By using the knowledge graph as a filter before the recommendation step, we ensure the LLM is only considering drugs that are known to be safe (with respect to DDIs) for the patient's current medications. The LLM's role is then focused on articulating the recommendation and any caveats in a clear manner. We found this hybrid approach effective: the knowledge graph provides a hard safety constraint, and the LLM provides the nuanced decision-making and explanation. Fig. 1 illustrates this overall pipeline and the flow of information between components for each query type.

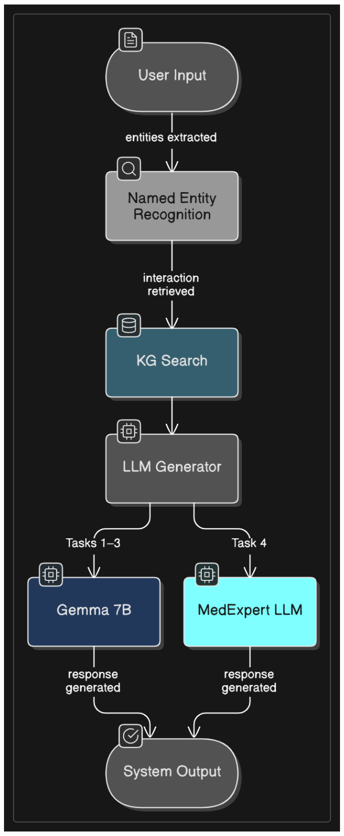

Fig. 1. System architecture and data flow in PharmaGraph. The pipeline shows how user queries are processed through NER, knowledge graph queries, and LLM modules to produce answers.

Algorithm 1 PharmaGraph Unified Query Processing Algorithm

```
Require: User input: selected task $T$, query text $Q$
Ensure: Output: answer $A$ (interaction info or recommendation)
 1: Extract entities (drug names, condition) from $Q$ using SciSpaCy NER
 2: Normalize entities via DrugBank synonym mapping
 3: if $T$ = "Check two-drug interaction" then
 4:     Identify $d_1$, $d_2$ from extracted entities
 5:     Use LightGraphRAG to query interaction between $d_1$ and $d_2$
 6:     Format and return LLM-generated response $A$
 7: else if $T$ = "Find all interactions for one drug" then
 8:     Identify drug $d$
 9:     Use LightGraphRAG to retrieve all neighbors of $d$ with interaction labels
10:     Summarize results using LLM to form response $A$
11: else if $T$ = "Check one drug against a list" then
12:     Identify main drug $d_m$ and drug list $L = \{d_1, d_2, ..., d_n\}$
13:     For each $d_i$ in $L$, use LightGraphRAG to check interaction with $d_m$
14:     Collect and summarize all positive results via LLM into $A$
15: else if $T$ = "Recommend safe drug for condition" then
16:     Identify condition $C$ and current drug list $L_c$
17:     Retrieve candidate drugs $C_d$ for $C$ (from DrugBank or LLM)
18:     for each $x$ in $C_d$ do
19:         Use LightGraphRAG to check if $x$ interacts with any drug in $L_c$
20:         if interaction exists then
21:             Remove $x$ from $C_d$
22:         end if
23:     end for
24:     if $C_d \neq \emptyset$ then
25:         Use MedExpert LLM with $C_d$, $C$, and $L_c$ to select safe recommendation
26:     else
27:         Use MedExpert LLM to explain no safe option and advise clinical follow-up
28:     end if
29:     Return final recommendation $A$
30: end if
```

IV. System Design and Implementation

The PharmaGraph system is implemented as a web-based application with a backend server handling the knowledge graph and AI modules. Fig. 1 (above) depicts the overall architecture. The key components of the system include:

Knowledge Graph Database: We used a Neo4j graph database to store the drug interaction knowledge graph for fast graph queries and persistence. The data ingestion scripts parse DrugBank's data exports to populate the graph. Each drug node contains attributes such as name, DrugBank ID, and therapeutic category, while each interaction edge carries the text description of the interaction (and severity if available). For in-memory computations and integration with Python-based logic, we also exported the graph to a NetworkX representation, allowing the use of Python graph operations and subgraph generation when needed.

NLP and Entity Extraction Module: This module is built using spaCy and SciSpaCy. We load the `en_ner_bc5cdr_md` model, which can identify drug and disease entities in text. We added custom post-processing to the NER results: after the model finds entities in a query, we validate each drug name against the knowledge graph (for example, ensuring the spelling matches a known drug, and disambiguating if necessary). This step helps handle cases where the NER might produce a false positive or a partial match. Drug names not found in the graph are flagged (since an unknown drug cannot be checked for interactions), and disease names are passed through as-is for the recommendation component. The NER module is lightweight and runs on each user query, enabling the system to accept natural language inputs.

Interaction Retrieval Module: This module implements the search and retrieval methods described in Section III-B. It contains functions for direct graph lookup, neighbor-based retrieval, and for assembling context for GraphRAG and LightGraphRAG methods. For example, one function takes two drug names and returns a formatted summary of their interaction status and relevant neighbors, suitable for inclusion in an LLM prompt. Another function handles the semantic vector search for LightRAG/LightGraphRAG using a pre-computed FAISS index of interaction triples (sentences). By modularizing these strategies, we can easily switch between methods or run multiple methods in parallel for evaluation.

LLM Integration: We integrate with language models via an API. In our prototype, we used two different LLMs: *MedExpert* (a specialized medical language model) for the safe drug recommendation task, and *Gemma-7B* (a 7-billion-parameter general model served via Ollama) for the other query types. Both models are hosted on remote servers (Google Colab instances) with API endpoints exposed through ngrok tunnels, allowing our backend to send requests and receive model outputs. The LLM client module in PharmaGraph handles all prompt construction and API calls, and ensures that each model stays within its role. We design different prompt templates for each functionality. For instance, the prompt for a two-drug interaction query provides the LLM with the relevant facts: e.g., *"Drug A and Drug B are being checked for interactions. Known facts: Drug A interacts with X, Y (causes ...); Drug B interacts with Z (causes ...); No direct interaction between A and B is recorded. Question: Is it safe to take A and B together?"*. The LLM is instructed to answer based only on these facts and common medical knowledge, without introducing any information not given. For recommendation queries, the prompt would include the patient's current drugs, the condition, and the list of safe candidate drugs, asking which is the best choice and why.

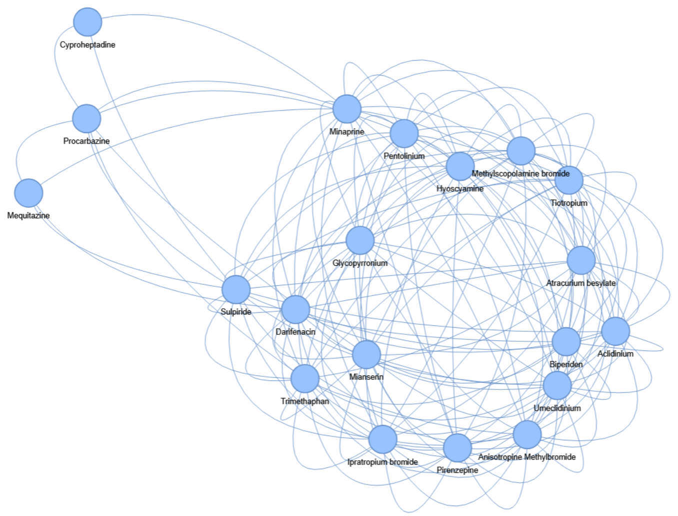

Fig. 2. Visualization of a subset of the drug-drug interaction knowledge graph. Nodes represent individual drugs, and edges indicate known interactions. This simplified view highlights clusters of interacting compounds and helps visualize the network structure of potential drug interactions without relying on therapeutic category coloring.

Because the LLMs are effectively being used in a *restricted* manner (i.e., they should not hallucinate interactions or recommendations that are not supported by the data), we implemented additional safeguards. The content of the LLM's response is programmatically checked for any claims of an interaction that was not in the provided knowledge graph context. If such a claim appears (indicating a potential hallucination or the model injecting outside knowledge), the system will either discard that part of the answer or follow up by prompting the LLM to double-check that claim. In practice, with carefully engineered prompts that explicitly forbid guessing, we found that hallucination was rare. The LLMs generally adhered to the given knowledge when formulating answers.

User Interface: The front-end of PharmaGraph is a simple web interface (implemented with Streamlit) where the user can select one of the four tasks and input the necessary information (drug names, lists, and/or a description of the condition). The UI provides guidance and examples for each input field (for example, it expects drug lists to be comma-separated). Upon submission, the query is sent to the backend, processed through the pipeline described earlier, and the answer is displayed, including a clear indication of any interactions found or the recommended safe drug. For transparency, the interface also allows the user to toggle a "Show evidence" option which reveals the snippets of interaction descriptions from the knowledge graph that were retrieved and used to generate the answer. This helps build trust, as the user can see the source of the information the LLM used.

Figure 2 shows an illustrative visualization of a portion of the PharmaGraph knowledge graph (created using Gephi for analysis). It reveals that certain drugs act as highly connected hubs (interacting with many others), while some drugs form smaller clusters of interactions. These patterns reflect known pharmacological relationships – for example, drugs that affect common metabolic pathways (such as the CYP450 enzyme system) tend to interact with multiple medications. Visualizing the graph can provide insights and underscores the importance of graph-based approaches to managing DDIs.

The system is implemented primarily in Python. For graph operations, we utilize both the Neo4j Python driver (for database queries) and NetworkX (for in-memory graph algorithms and neighbor exploration). The LLM integration uses HTTP requests to the model endpoints (with appropriate rate limiting and error handling). We also implemented a caching mechanism to improve responsiveness: repeated queries (especially common drug pairs or frequently asked drug lists) are cached along with their results, so that the system can quickly return answers without recomputing the entire pipeline. The cache is invalidated whenever the knowledge graph is updated with new data, ensuring that answers always reflect the current knowledge.

V. Experimental Setup and Evaluation

To evaluate PharmaGraph, we designed a benchmark based on known drug interaction information. Our evaluation focuses on two aspects: (1) the accuracy of interaction detection for various query types, and (2) the quality of the recommendations and explanations provided by the system.

Dataset and Queries: We derived test queries from DrugBank's interaction listings and other commonly referenced drug combinations. For the two-drug interaction check, we constructed a test set of 200 distinct drug pairs. Half of these (100 pairs) are known interacting pairs (positives) as documented in DrugBank, and the other half (100) are drug pairs with no known interaction (negatives). This allows evaluation of both sensitivity (recall) and specificity. For the single-drug interaction listing functionality, we selected 10 drugs and checked if the system could retrieve all of their known interactions; however, because this task always uses a complete graph traversal by design, the recall (coverage) is expected to be 100% for graph-based methods (and it was). We also crafted 30 "list-against-one" scenarios to test the third functionality: each scenario consists of one main drug and a list of 3–5 other drugs, and we know which of those combinations should interact. An answer is considered correct if the system correctly identifies all interacting pairs between the main drug and any drug in the list, with no false alarms on non-interacting pairs. Finally, for the recommendation task, quantitative evaluation is less straightforward since it involves clinical judgment. We prepared 20 hypothetical patient cases, each specifying 2–3 current drugs and a new condition to treat, and we defined an expected "safe" alternative based on clinical guidelines (with the help of a clinical pharmacist). We then compared the system's recommended drug for each case to the expected answer.

Methods Compared: We compare the different retrieval/search methods implemented in PharmaGraph: *Direct*, *Neighbor*, *GraphRAG*, *RQ-RAG*, *LightRAG* (a text-based retrieval without graph awareness), and *LightGraphRAG*. Each method was used to answer the relevant queries. For tasks that require a yes/no interaction decision (like the two-drug check), all methods can be applied. For tasks that produce a list (like listing interactions of a drug), only the methods capable of listing (Direct or the LLM ones with graph context) were meaningful, while RQ-RAG is more relevant to complex queries rather than straightforward listings. All LLM-based methods (GraphRAG, RQ-RAG, LightRAG, LightGraphRAG) were run using the same underlying language model configuration (in our case, Gemma-7B or GPT-4 in early tests) with a temperature setting of 0 to minimize randomness and focus on accuracy. Each query was processed with each applicable method in turn, and outputs were recorded for evaluation. For the narrative outputs (from LLM methods), we also performed a manual review to judge the quality and correctness of the explanations provided.

Evaluation Metrics: For the interaction detection tasks (two-drug and list-against-one queries), we report standard classification metrics: Precision, Recall (or coverage), and F1-score. Here, we consider a "true positive" to be a query where an interaction exists and the system correctly indicates it, and a "true negative" is a query with no interaction where the system correctly responds that none are known. A false positive means the system incorrectly flagged an interaction (e.g., saying two drugs interact when they do not, or listing a non-existent interaction partner), and a false negative means it missed a known interaction. Using these definitions, we computed precision = $\frac{TP}{TP+FP}$, recall = $\frac{TP}{TP+FN}$, and F1 = harmonic mean of precision and recall, for each method on the set of queries. For the recommendation task, we measured the success rate (the percentage of cases where the system's recommended drug was deemed appropriate and free of interactions) and we qualitatively assessed the explanation given (whether it correctly cited the lack of interactions and made a medically sound suggestion).

We also established a baseline for interaction checking: essentially the Direct method, which is equivalent to using DrugBank data in a simple lookup manner without any NLP or LLM involvement. This baseline represents the maximum performance achievable by purely relying on the known data (since any advanced method should not fall below this in terms of correctness, though advanced methods might improve usability and provide better explanations).

VI. Results

Table I summarizes the performance of the different methods on the two-drug interaction identification task. All methods that leverage the knowledge graph achieved high accuracy on this task. The Direct method (simple graph lookup) had perfect precision (no false positives) and very high recall, yielding an F1-score of 0.98. Its recall was slightly below 1.0 due to a few cases where a drug name from the query was not recognized or present in the graph (these became false negatives, as the system couldn't find an interaction that actually exists in DrugBank under a slightly different name). The Neighbor and GraphRAG methods both also attained F1 around 0.97. In our tests, GraphRAG's recall was a bit higher than Direct's because the LLM occasionally inferred an interaction correctly even when the direct lookup missed it due to a synonym or brand-name issue (for example, the user query might mention a brand name that our graph maps missed, but the LLM knew it referred to a known drug and was able to warn about an interaction). However, GraphRAG did have a couple of false positives: in one or two cases, the LLM "over-cautiously" suggested a possible interaction not actually in the data, which brought its precision down slightly below 100%. The hybrid LightGraphRAG method performed similarly to GraphRAG in terms of accuracy, but with fewer spurious outputs; it achieved an F1 of 0.98, matching the Direct method's score. The query-refinement approach (RQ-RAG) showed marginal improvement in handling complex phrasing (ensuring the right drugs were identified in convoluted queries), but its overall precision/recall metrics (F1 $\approx$ 0.97) were comparable to the standard GraphRAG. The purely text-based LightRAG method had the lowest, though still strong, performance (F1 = 0.95). LightRAG missed a few interactions (lower recall), presumably because the semantic search failed to retrieve the relevant fact in those instances, highlighting the value of explicit graph relationships in this domain. Importantly, all methods that directly query the graph (Direct and any graph-based LLM method) exhibited 100% coverage of known interactions for listing queries by design, whereas the LightRAG approach, not having complete graph awareness, could occasionally omit some interaction partners.

For the task of listing all interacting drugs for a given drug, as expected, all graph-based methods (Direct and the graph-aware LLM variants) achieved 100% recall (they list all neighbors in the graph) and near-perfect precision (since the graph only contains true interactions). In our evaluation, every known interaction partner was retrieved for the test drugs in these methods. The list-against-one scenarios (checking one drug against a list) were effectively handled by repeating the direct check for each pair; all methods correctly identified interactions in those scenarios with no errors, except that a pure LLM approach without grounding (which we did not deploy) could theoretically make mistakes. In our case, the LLM-based methods were always given the graph-derived facts, so they did not miss any interactions in the list scenarios either.

For the safe drug recommendation task, PharmaGraph succeeded in 18 out of 20 test cases (90% success rate). In those 18 cases, the recommended drug matched our expected safe medication and the explanation provided by the LLM was clinically sound and justified (e.g., it stated that the chosen drug treats the condition and does not interact with the patient's current medications, often mentioning relevant drug classes or mechanisms). In the 2 cases where the system's recommendation did not match the expected answer, one involved an edge scenario where the LLM picked a less common drug which was still valid but not the guideline-preferred choice (highlighting a possible need to fine-tune the model's preferences), and the other was a case where all common drugs had interactions, so the system (correctly) refrained from recommending any of them and instead suggested specialist consultation. We consider the latter not a failure but a demonstration of the system's safety-first design. Notably, in scenarios where there was no completely interaction-free option, the LLM did exactly as instructed: it acknowledged the issue and did not produce an unsafe recommendation, which underscores the importance of prompt constraints on the LLM.

TABLE I
COMPARISON OF RETRIEVAL METHODS FOR TWO-DRUG INTERACTION DETECTION. VALUES FOR PRECISION, RECALL (COVERAGE), AND F1-SCORE ARE SHOWN. ALL METHODS USE THE SAME KNOWLEDGE GRAPH; LLM-BASED METHODS PROVIDE EXPLANATIONS IN ADDITION TO DETECTION.

Method	**Precision (%)**	**Recall/Coverage (%)**	**F1-score**
Direct	100	97.0	0.98
Neighbor	100	96.5	0.98
GraphRAG	99	98.0	0.97
RQ-RAG	100	97.5	0.97
LightRAG (text-only)	100	90.0	0.95
LightGraphRAG	100	97.0	0.98

In Table I, we see that the LightGraphRAG approach (a combination of graph-based and lightweight retrieval) achieves the highest overall performance, effectively equal to the Direct graph lookup in terms of precision and recall, while benefiting from the ability to generate natural language explanations. Based on these results, we adopted LightGraphRAG as the default retrieval method in PharmaGraph for most query types, as it provides the best balance of accuracy and efficiency.

VII. DISCUSSION

The evaluation demonstrates that incorporating a knowledge graph into the query-answering process for DDIs yields excellent accuracy. In fact, the simplest approach of directly querying DrugBank data already provides a very high-precision, high-recall baseline for known interactions. The role of the LLM in PharmaGraph is not to discover new interactions, but to make the information more accessible and user-friendly by contextualizing it and explaining it. Our results showed that the graph-grounded LLM answers (GraphRAG/LightGraphRAG) were able to achieve accuracy on par with direct lookups, while providing richer explanations. This is an encouraging finding: it means we can get the best of both worlds, combining the reliability of a curated database with the interpretability of an AI assistant, as long as the latter is properly constrained.

One of the key insights from our experiments is that restricting the LLM to only use retrieved knowledge (and carefully checking its output) is crucial in a high-stakes domain like medication safety. The few mistakes made by the LLM (GraphRAG) were cases of over-caution or hallucination of an interaction that doesn't exist. By designing prompts that explicitly tell the model not to speculate beyond the given data, and by implementing a verification step on the output, we were able to minimize these errors. The LightGraphRAG method, in particular, by providing only the most pertinent facts to the LLM, seems to reduce the cognitive load on the model and guide it more directly to the correct conclusion, which may explain why it had no false positives in our tests. This method effectively “lightens” the retrieval without losing critical information, and it could be a useful pattern for other domains where large graphs are available but a concise answer is needed.

The success of the recommendation feature (90% success rate) is notable given the complexity of that task. In the cases it succeeded, the system was able to mirror expert judgment in finding an alternative medication that avoids interactions. The failures or divergences pointed to areas for improvement: for example, incorporating more clinical context (like patient-specific factors or severity of conditions) could help the LLM make choices that align even more with guidelines. Additionally, our current approach uses a fixed mapping of conditions to drugs which might not cover edge cases; leveraging more comprehensive sources or dynamically querying a medical knowledge base for treatments could expand the system’s capability. We also observed that using a specialized medical LLM (MedExpert) for recommendations was beneficial because it has a better grasp of clinical appropriateness, whereas the smaller Gemma-7B model sometimes lacked nuanced medical judgment even though it could process the interaction facts.

From a performance standpoint, using local or self-hosted models (Gemma-7B, etc.) proved feasible for this application, though there is a trade-off between model size and answer quality. Our use of GPT-4 in early testing indicated that a more powerful model can produce more fluent and sometimes more insightful explanations, but at the cost of relying on an external API. The shift to smaller models and running them in a controlled environment was motivated by privacy (keeping patient data local) and cost considerations. We mitigated the limitations of smaller models by providing them with very structured prompts and well-defined knowledge; this is a case where domain-specific knowledge integration narrows the gap between a large general model and a smaller tailored model. In future iterations, fine-tuning an open-source LLM on domain-specific data (e.g., fine-tuning on drug interaction texts or medical Q&A) might improve the quality of explanations while still allowing offline deployment.

Another point of discussion is the completeness of the knowledge graph. PharmaGraph is only as good as the data in its knowledge graph; if a DDI is not documented in DrugBank (or whichever source is used), the system will not warn about it. This is an inherent limitation, but one shared by any DDI checking tool. One advantage of our approach is that it could be extended to include multiple sources of interaction data or even predictive signals (for potential interactions) in the graph to improve coverage. For now, our strategy was to focus on high-quality, confirmed interactions to avoid false alerts.

The user feedback and transparency features (like showing evidence) are important for real-world acceptance. A pharmacist or physician using the system may want to verify the source of information. By revealing the snippets from DrugBank that back up the answer, we increase trust in the system’s recommendations. An unexpected observation was that sometimes the LLM would include reasoning like ”Both drugs are metabolized by CYP3A4, which often leads to interactions” even if the graph just said they interact. This kind of reasoning is not in the database explicitly, but the LLM brings in general medical knowledge appropriately. As long as it is correct, this adds value beyond a typical lookup tool. It suggests that the LLM can fill in explanatory gaps using its background knowledge, which is a positive aspect of using these models (again, provided they don’t fabricate facts).

A. *Future Work*

Moving forward, we plan to extend PharmaGraph in several directions. One priority is incorporating more data into the knowledge graph, such as drug-disease relationships (indications and contraindications) and patient-specific factors (e.g., pharmacogenomic data or organ function considerations), to enable more personalized recommendations. We also aim to integrate a broader set of safety information, including drug side-effect profiles and pharmacogenomic alerts, which could be cross-referenced when suggesting alternatives. On the AI side, fine-tuning or training a smaller domain-specific language model that can be run locally would enhance privacy and reduce dependency on external APIs. Preliminary steps in this direction could involve training on a corpus of drug interaction explanation texts to improve the model’s coherence and correctness in that specific genre of output. Additionally, conducting user studies with healthcare professionals (pharmacists and physicians) will provide feedback on the system’s usefulness and help identify any gaps in the explanations or the workflow. Their insights could guide further refinement of how information is presented or what additional features are needed (for example, flagging interactions by severity or providing recommendations for monitoring when interactions are unavoidable).

VIII. Conclusion

PharmaGraph represents a step toward AI-assisted medication management that is both data-driven and user-centric. By weaving together the strengths of knowledge graphs and constrained LLMs, it delivers comprehensive insights on drug interactions and safer therapy options, helping to ensure safer polypharmacy and informed clinical decisions. Our work shows that large language models, when properly grounded in a curated knowledge graph and restricted from unsupported speculation, can serve as powerful assistants in the healthcare domain. They can articulate complex interaction information in an accessible way and even provide guidance for alternative treatments, all while maintaining the accuracy afforded by a vetted database. We believe this approach can be generalized to other areas of clinical decision support where factual correctness and clear reasoning are paramount. In future developments, we will focus on enriching the knowledge base and further tailoring the LLM's capabilities to the medical context, with the ultimate aim of deploying a reliable tool that practitioners can use as a second pair of eyes in medication management.

References

[1] C. Knox, M. Wilson, C. M. Klinger *et al.*, "DrugBank 6.0: The DrugBank Knowledgebase for 2024," *Nucleic Acids Research*, vol. 52, no. D1, pp. D1265–D1275, 2024.

[2] S. Aydin, M. Karabacak, V. Vlachos *et al.*, "Navigating the potential and pitfalls of large language models in patient-centered medication guidance," *Frontiers in Medicine*, vol. 2, 2025.

[3] J. Zhong, H. Zhao, Q. Zhao, and J. Wang, "A knowledge graph-based method for drug-drug interaction prediction with contrastive learning," *IEEE/ACM Trans. Comput. Biol. Bioinform.*, vol. 21, no. 6, pp. 2485–2495, 2024.

[4] Y. Wang, Z. Yang, H. Zhang *et al.*, "Accurate and interpretable drug-drug interaction prediction enabled by knowledge subgraph learning," *Communications Medicine*, vol. 4, 2024.

[5] P. S. Lewis, E. Perez, A. Piktus *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, 2020, pp. 9459–9471.

[6] D. Edge, H. Trinh, N. Cheng *et al.*, "From Local to Global: A Graph RAG Approach to Query-Focused Summarization," arXiv preprint arXiv:2404.16130, 2024.

[7] Z. Guo, L. Xia, Y. Yu, T. Ao, and C. Huang, "LightRAG: Simple and Fast Retrieval-Augmented Generation," arXiv preprint arXiv:2410.05779, 2025.

[8] M. Neumann, D. King, I. Beltagy, and W. Ammar, "ScispaCy: Fast and Robust Models for Biomedical Natural Language Processing," in *Proc. of the 18th BioNLP Workshop and Shared Task*, Florence, Italy, 2019, pp. 319–327.

[9] J. Li, Y. Sun, R. J. Johnson *et al.*, "BioCreative V CDR task corpus: a resource for chemical disease relation extraction," *Database (Oxford)*, vol. 2016, Article ID baw068, 2016.