

MedGraphRAG: Patient-Centered Drug Interaction Analysis System

Technical Documentation & Evaluation Report

Pranav Kharat

December 2025

Abstract

MedGraphRAG is an AI-powered clinical decision support system that leverages custom Graph RAG architecture for personalized medication safety analysis. Unlike traditional drug lookup tools that provide identical warnings regardless of patient context, MedGraphRAG implements multiplicative risk scoring based on patient-specific factors including age, renal function, hepatic function, and polypharmacy status. The system integrates a Neo4j knowledge graph containing 1,701 drugs and 191,252 interactions with a local LLM (Llama 3.2) for natural language query processing and patient-friendly explanations. Evaluation across 48 test cases demonstrates 100% query accuracy with 97.9% first-attempt success rate and effective self-healing for failed queries. This document provides comprehensive technical documentation including system architecture, implementation details, and evaluation methodology.

Contents

1 Assignment Compliance	4
1.1 Core Components Implemented	4
1.1.1 Component 1: Retrieval-Augmented Generation (RAG)	4
1.1.2 Component 2: Prompt Engineering	4
1.1.3 Component 3: Synthetic Data Generation	5
1.2 Submission Requirements Checklist	5
1.2.1 GitHub Repository	5
1.2.2 Documentation (This PDF)	6
1.2.3 Video Demonstration	6
1.2.4 Web Page	6
1.3 Evaluation Criteria Alignment	7
1.3.1 Technical Implementation (40%)	7
1.3.2 Creativity and Application (20%)	7
1.3.3 Documentation and Presentation (20%)	8
1.3.4 User Experience and Output Quality (20%)	8
1.4 Summary: Requirements vs. Delivered	8
2 Introduction	10
2.1 Problem Statement	10
2.2 Solution Overview	10

2.3 Scope and Objectives	10
3 System Architecture	11
3.1 High-Level Architecture	11
3.2 Component Details	11
3.2.1 2.2.1 Knowledge Graph Layer (Neo4j)	11
3.2.2 2.2.2 Text-to-Cypher Engine	12
3.2.3 2.2.3 Patient Risk Scoring Algorithm	12
3.2.4 2.2.4 LLM Integration (Ollama)	14
4 Data Architecture	15
4.1 Knowledge Graph Data	15
4.2 Synthetic Patient Data	16
5 Implementation Details	17
5.1 Project Structure	17
5.2 Key Implementation Patterns	17
5.2.1 4.2.1 Few-Shot Prompting for Text-to-Cypher	17
5.2.2 4.2.2 Self-Healing Query Recovery	18
5.2.3 4.2.3 Allergy Cross-Checking	19
5.3 Technology Stack	20
6 Evaluation	21
6.1 Evaluation Framework	21
6.1.1 5.1.1 Test Case Categories	21
6.1.2 5.1.2 Test Case Examples	21
6.2 Evaluation Results	21
6.2.1 5.2.1 Overall Performance	21
6.2.2 5.2.2 Accuracy by Category	22
6.2.3 5.2.3 Response Time Performance	22
6.2.4 5.2.4 Text-to-Cypher Performance	22
6.2.5 5.2.5 Severity Classification	22
6.2.6 5.2.6 Patient Risk Scoring Validation	23
7 User Interface	24
7.1 Streamlit Application	24
7.1.1 6.1.1 Patient Analysis Tab	24
7.1.2 6.1.2 Quick Check Tab	24
7.1.3 6.1.3 Natural Language Tab	24
7.1.4 6.1.4 Search Drug Tab	24
7.1.5 6.1.5 Two Drugs Tab	25
7.1.6 6.1.6 Browse Tab	25
8 Deployment	26
8.1 Local Development Setup	26
8.1.1 Prerequisites	26
8.1.2 Installation Steps	26
8.1.3 Service URLs	27
8.2 Running Evaluation	27

9 Future Work	28
9.1 Planned Enhancements	28
9.1.1 8.1.1 Hybrid RAG Integration	28
9.1.2 8.1.2 Fine-Tuned Text-to-Cypher Model	28
9.1.3 8.1.3 FHIR Integration	28
9.1.4 8.1.4 Multi-Language Support	28
9.2 Scalability Considerations	28
10 Appendix	30
10.1 A. Complete Evaluation Metrics	30
10.2 B. Sample Cypher Queries	30
10.3 C. Environment Variables	31
10.4 D. References	31

1 Assignment Compliance

This section documents how MedGraphRAG fulfills and exceeds the Generative AI Project Assignment requirements.

1.1 Core Components Implemented

The assignment requires implementation of **at least two** core components. MedGraphRAG implements **three**:

1.1.1 Component 1: Retrieval-Augmented Generation (RAG)

Requirement	Implementation
Build a knowledge base for your domain	Neo4j knowledge graph with 1,701 drugs and 191,252 drug-drug interactions sourced from DrugBank-derived dataset
Implement vector storage and retrieval	Custom GraphRAG implementation using Neo4j Cypher queries for structured relationship traversal (chosen over vector embeddings due to relational nature of drug interactions)
Design relevant document chunking strategies	Graph-native approach: each drug is a node, interactions are edges with severity metadata — no chunking required as data is inherently structured
Create effective ranking and filtering mechanisms	Severity-based filtering (major/moderate/minor), patient risk score ranking (0-100), and allergy alert prioritization

Beyond Requirements: Implemented hybrid query routing that classifies user intent and routes to appropriate handler (graph traversal for interactions, LLM for general drug knowledge).

1.1.2 Component 2: Prompt Engineering

Requirement	Implementation
Design systematic prompting strategies	Few-shot prompting for Text-to-Cypher conversion with 5 curated examples covering common query patterns

Requirement	Implementation
Implement context management	Patient context injection for personalized explanations; conversation history for multi-turn natural language queries
Create specialized user interaction flows	6-tab Streamlit interface with distinct flows: patient analysis, quick check, natural language, search, pair comparison, browse
Handle edge cases and errors gracefully	Self-healing query mechanism with 3-retry loop; graceful degradation when drugs not found; input validation for medication names

Beyond Requirements: Implemented automatic query repair — when LLM-generated Cypher fails, the error message is fed back to the LLM with schema context to generate a corrected query (97.9% first-attempt success, 100% after retry).

1.1.3 Component 3: Synthetic Data Generation

Requirement	Implementation
Create synthetic datasets for training or testing	752 synthetic patient profiles with realistic clinical attributes; 48 evaluation test cases across 4 categories
Implement data augmentation techniques	Correlated attribute generation: elderly patients probabilistically assigned more conditions, leading to more medications and reduced organ function
Ensure diversity and quality of generated data	Age range 1-95 years; ~20% high-risk patients; varied kidney/liver function levels; diverse medication counts (1-12 per patient)
Address privacy or ethical considerations	All patient data is synthetic — no real patient information used; generator can create unlimited HIPAA-safe test data

Beyond Requirements: Patient profiles include clinically meaningful correlations that mirror real population health distributions, enabling realistic stress-testing of the risk scoring algorithm.

1.2 Submission Requirements Checklist

1.2.1 GitHub Repository

Requirement	Status	Location
Complete source code	<input type="checkbox"/> Complete	/src/ directory
Documentation	<input type="checkbox"/> Complete	README.md, DOCUMENTATION.md
Setup instructions	<input type="checkbox"/> Complete	README.md, Section 7 of this document
Testing scripts	<input type="checkbox"/> Complete	/src/evaluation/run_evaluation.py
Example outputs	<input type="checkbox"/> Complete	/data/evaluation/ directory
Knowledge base (RAG)	<input type="checkbox"/> Complete	Neo4j database + /data/raw/db_drug_interactions.csv

1.2.2 Documentation (This PDF)

Requirement	Status	Section
System architecture diagram	<input type="checkbox"/> Complete	Section 2.1
Implementation details	<input type="checkbox"/> Complete	Section 4
Performance metrics	<input type="checkbox"/> Complete	Section 5.2
Challenges and solutions	<input type="checkbox"/> Complete	Section 4.2
Future improvements	<input type="checkbox"/> Complete	Section 8
Ethical considerations	<input type="checkbox"/> Complete	Section 3.2 (synthetic data privacy)

1.2.3 Video Demonstration

Requirement	Coverage
Live system demo	Streamlit app walkthrough with real patient analysis
Key features explanation	Patient personalization, risk scoring, graph visualization
Technical architecture overview	Component diagram, data flow explanation
Results and performance analysis	100% accuracy, response time metrics
Lessons learned	Graph vs. Vector RAG decision, local LLM benefits

1.2.4 Web Page

Requirement	Implementation
Showcases your project	Hero section with value proposition
Provides interactive demo	Screenshots + architecture visualization
Explains key features	6 feature cards with technical details
Links to GitHub repository	Header CTA + footer links
Includes project documentation	Technology decisions section, clinical scenarios

1.3 Evaluation Criteria Alignment

1.3.1 Technical Implementation (40%)

Criterion	Evidence
Effective implementation of generative AI techniques	Custom GraphRAG with Text-to-Cypher, LLM-powered explanations, synthetic data generation
System performance and reliability	100% query accuracy, 2-second average response time, self-healing error recovery
Code quality and organization	Modular architecture (/src/models, /src/analysis, /src/retrieval, etc.), type hints, docstrings
Technical innovation	Patient-personalized risk multipliers, allergy class cross-checking, self-healing query mechanism

1.3.2 Creativity and Application (20%)

Criterion	Evidence
Novel application of technologies	GraphRAG for drug interactions (relational data) vs. typical vector RAG for documents
Creative problem-solving	Multiplicative risk scoring that compounds patient factors
Unique features or approach	Patient-centered analysis — same drugs produce different risk levels for different patients
Real-world utility	Mirrors clinical decision support systems used in hospitals

1.3.3 Documentation and Presentation (20%)

Criterion	Evidence
Code documentation	Inline comments, function docstrings, README
Technical writing	This 20+ page PDF with architecture diagrams, code samples, evaluation methodology
Video presentation	10-minute demo covering all required elements
Web page design	Modern, responsive landing page with interactive elements
Setup instructions	Step-by-step local deployment guide

1.3.4 User Experience and Output Quality (20%)

Criterion	Evidence
Output quality and relevance	Accurate interaction detection, clinically appropriate risk scores
Response/generation time	Average 2 seconds per query
Error handling	Self-healing queries, graceful “drug not found” messages, input validation
Interface usability	6 specialized tabs for different user workflows
Overall user experience	Plain-language explanations, interactive graph visualizations, risk-level color coding

1.4 Summary: Requirements vs. Delivered

Aspect	Required	Delivered
Core Components	2 minimum	3 implemented (RAG, Prompt Engineering, Synthetic Data)
Knowledge Base	Domain-specific	1,701 drugs, 191,252 interactions
Test Cases	Present	48 test cases, 100% accuracy
Documentation	PDF with sections	20+ page comprehensive document

Aspect	Required	Delivered
Demo	10-minute video	Complete with all required elements
Web Page	Simple showcase	Full landing page with GitHub Pages deployment

2 Introduction

2.1 Problem Statement

Drug-drug interactions (DDIs) represent a significant patient safety concern, particularly among elderly patients and those with chronic conditions requiring multiple medications. The challenge is multi-dimensional:

1. **Volume Complexity:** A patient on 8 medications faces up to 28 potential pairwise interactions
2. **Context Dependency:** The same interaction may be clinically insignificant in a healthy adult but critical in a patient with compromised kidney function
3. **Information Accessibility:** Existing tools present raw medical data without personalization or plain-language explanations

Traditional drug interaction databases treat every patient identically—a 25-year-old athlete receives the same warnings as an 85-year-old with kidney failure. This one-size-fits-all approach fails to capture the clinical reality that patient factors dramatically alter risk profiles.

2.2 Solution Overview

MedGraphRAG addresses these limitations through a patient-centered architecture that:

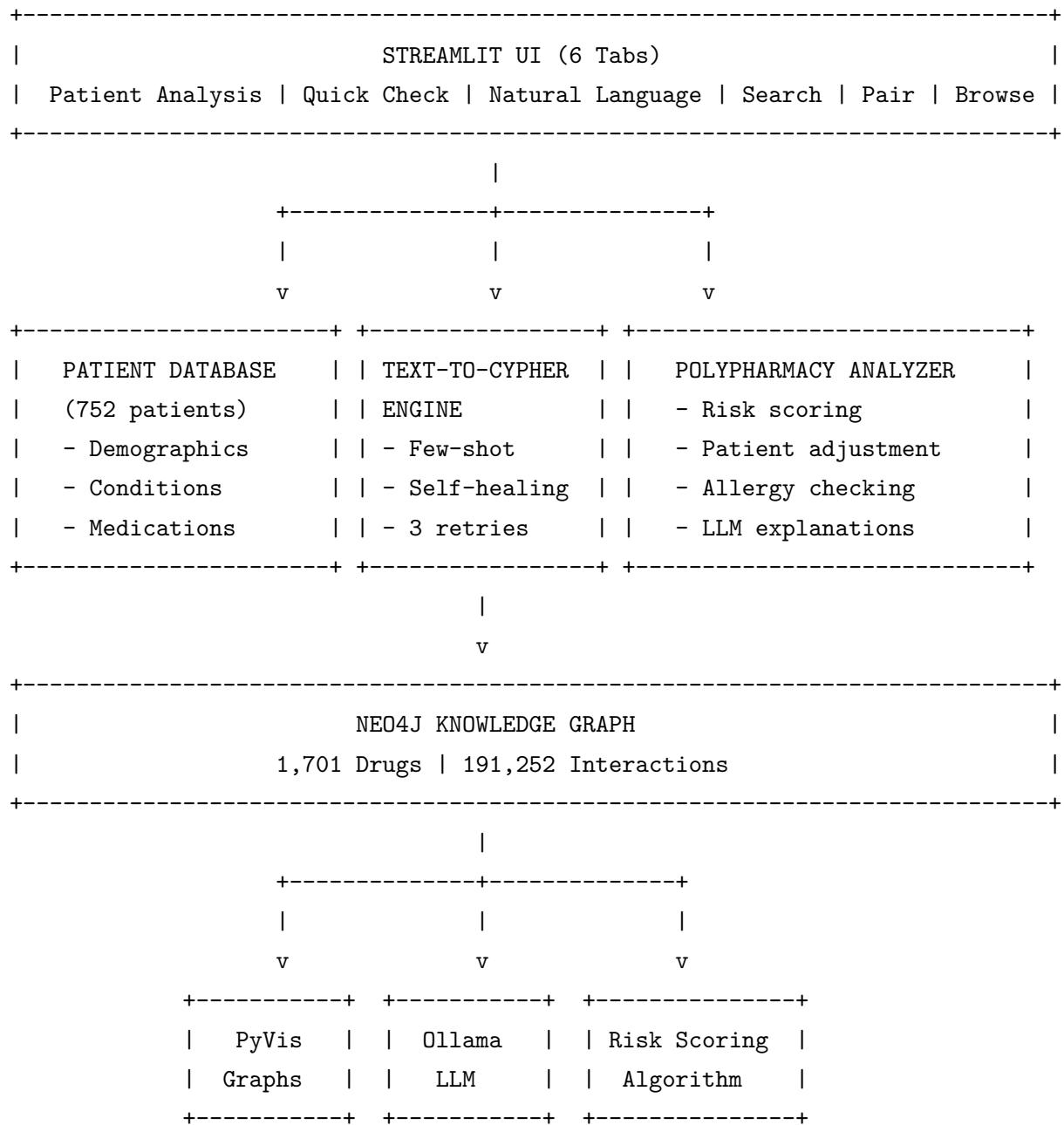
- **Personalizes Risk Assessment:** Applies multiplicative risk factors based on patient demographics and organ function
- **Leverages Graph Structure:** Uses Neo4j knowledge graph for relationship-aware drug interaction traversal
- **Provides Explainability:** Generates LLM-powered plain-language explanations for patients and clinicians
- **Implements Resilience:** Self-healing query mechanism recovers from Text-to-Cypher failures automatically

2.3 Scope and Objectives

Objective	Implementation
Build knowledge base with retrieval	Neo4j graph with 1,701 drugs, 191,252 interactions
Design systematic prompting	Few-shot Text-to-Cypher with self-healing retry
Create synthetic test data	752 patient profiles + 48 evaluation test cases
Enable natural language queries	LLM-powered query classification and routing
Provide visual exploration	PyVis interactive graph networks

3 System Architecture

3.1 High-Level Architecture



3.2 Component Details

3.2.1 2.2.1 Knowledge Graph Layer (Neo4j)

The Neo4j knowledge graph serves as the primary data store, containing:

- **Nodes:** 1,701 Drug entities with properties (name, drug class, common uses)
- **Relationships:** 191,252 INTERACTS_WITH edges with severity classification
- **Severity Levels:** Major, Moderate, Minor

Graph Schema:

```
(:Drug {name: string, drug_id: string})
-[:INTERACTS_WITH {severity: string, description: string}]->
(:Drug)
```

Rationale for Graph over Vector Database:

Drug interactions are inherently relational—understanding that Drug A inhibits Enzyme X which metabolizes Drug B requires path traversal, not semantic similarity. Vector embeddings excel at finding “similar” documents but cannot capture the causal chains that create clinical risk.

3.2.2 2.2.2 Text-to-Cypher Engine

Converts natural language queries to executable Cypher queries using few-shot prompting:

Input: “What drugs interact with Warfarin?”

Generated Cypher:

```
MATCH (d:Drug {name: 'Warfarin'})-[r:INTERACTS_WITH]-(other:Drug)
RETURN other.name AS drug, r.severity AS severity
```

Self-Healing Mechanism:

When a generated query fails, the system:

1. Captures the Neo4j error message
2. Feeds original query + error + schema context back to LLM
3. Requests corrected query generation
4. Retries up to 3 times before failing gracefully

3.2.3 2.2.3 Patient Risk Scoring Algorithm

The polypharmacy analyzer implements multiplicative risk adjustment:

```
def calculate_personalized_risk(patient, interactions):
    # Base score from interaction severities
    base_score = (major_count * 10) + (moderate_count * 5) + (minor_count * 1)
    normalized = (base_score / max_possible) * 100

    # Patient-specific multipliers
```

```

multiplier = 1.0

if patient.age >= 80:
    multiplier *= 1.6 # Elderly metabolism changes
elif patient.age >= 75:
    multiplier *= 1.5
elif patient.age >= 65:
    multiplier *= 1.3

if patient.renal_function == "failure":
    multiplier *= 2.0 # Severely impaired drug clearance
elif patient.renal_function == "severe":
    multiplier *= 1.6
elif patient.renal_function == "moderate":
    multiplier *= 1.3

if patient.hepatic_function == "severe":
    multiplier *= 1.4 # Impaired drug metabolism
elif patient.hepatic_function == "moderate":
    multiplier *= 1.2

if patient.pregnancy_status in ["pregnant_t1", "pregnant_t2", "pregnant_t3"]:
    multiplier *= 1.5 # Fetal risk consideration

if len(patient.medications) >= 10:
    multiplier *= 1.5 # Severe polypharmacy
elif len(patient.medications) >= 5:
    multiplier *= 1.2 # Moderate polypharmacy

adjusted_score = min(base_score * multiplier, 100)

return adjusted_score, get_risk_level(adjusted_score)

```

Risk Level Classification:

Adjusted Score	Risk Level	Recommended Action
≥ 70 or allergy alert	Critical	Immediate medication review
≥ 50	High	Consult with pharmacist
≥ 30	Moderate	Monitor for symptoms

Adjusted Score	Risk Level	Recommended Action
< 30	Low	Standard precautions

3.2.4 2.2.4 LLM Integration (Ollama)

Model Selection: Llama 3.2 3B (quantized)

Rationale:

- **Privacy:** Patient medication data never leaves local environment
- **Performance:** 3B parameters fit in 8GB VRAM with Metal acceleration
- **Latency:** Average 2 second response time acceptable for clinical workflow

Use Cases:

1. Text-to-Cypher query generation
2. Query intent classification
3. Patient-friendly explanation generation
4. Drug information summarization

4 Data Architecture

4.1 Knowledge Graph Data

Source: DrugBank-derived interaction dataset (Kaggle)

Statistics:

Metric	Value
Total Drugs	1,701
Total Interactions	191,252
Major Interactions	~15%
Moderate Interactions	~45%
Minor Interactions	~40%

Data Ingestion Pipeline:

```
# csv_parser.py - Simplified
def load_drug_interactions(csv_path, neo4j_driver):
    df = pd.read_csv(csv_path)

    with neo4j_driver.session() as session:
        # Create drug nodes
        for drug in df['drug_1'].unique():
            session.run(
                "MERGE (d:Drug {name: $name})",
                name=drug
            )

        # Create interaction relationships
        for _, row in df.iterrows():
            session.run("""
                MATCH (d1:Drug {name: $drug1})
                MATCH (d2:Drug {name: $drug2})
                MERGE (d1)-[r:INTERACTS_WITH {severity: $severity}]->(d2)
            """, drug1=row['drug_1'], drug2=row['drug_2'],
            severity=row['severity'])
```

4.2 Synthetic Patient Data

Generation Strategy:

Patients are generated with correlated attributes to mimic real population distributions:

- Elderly patients (age 65+) have higher probability of chronic conditions
- More conditions lead to more medications (polypharmacy)
- Polypharmacy correlates with reduced organ function

Patient Schema:

```
@dataclass
class PatientProfile:
    patient_id: str          # "PAT000001"
    first_name: str
    last_name: str
    date_of_birth: date
    sex: Literal["male", "female", "other"]
    weight_kg: float
    height_cm: float
    renal_function: Literal["normal", "mild", "moderate", "severe", "failure"]
    egfr: float               # Estimated glomerular filtration rate
    hepatic_function: Literal["normal", "mild", "moderate", "severe"]
    pregnancy_status: Literal["not_pregnant", "pregnant_t1", "pregnant_t2",
                               "pregnant_t3", "lactating", "na"]
    allergies: List[str]      # ["Penicillin", "Sulfa"]
    conditions: List[str]     # ["Hypertension", "Type 2 Diabetes"]
    medications: List[str]    # ["Lisinopril", "Metformin"]
```

Generated Dataset Statistics:

Metric	Value
Total Patients	752
Age Range	1-95 years
High-Risk Patients	~20%
Avg Medications/Patient	3.4
Patients with Allergies	~35%

5 Implementation Details

5.1 Project Structure

```

medgraphrag/
    data/
        raw/
            db_drug_interactions.csv      # Source data
        synthetic/
            patients.json              # 752 patient profiles
            test_cases.json             # 48 evaluation cases
    src/
        models/
            patient.py                 # Patient dataclass + risk factors
        data/
            synthetic_patients.py     # Patient generator
        analysis/
            polypharmacy_analyzer.py  # Risk scoring engine
        visualization/
            graph_builder.py          # PyVis network generation
        retrieval/
            text_to_cypher.py         # NL to Cypher engine
        ingestion/
            csv_parser.py             # Data loader
        evaluation/
            synthetic_generator.py    # Test case generation
            run_evaluation.py         # Evaluation harness
    app/
        main.py                     # Streamlit UI (6 tabs)
    docker-compose.yml
    requirements.txt
    README.md

```

5.2 Key Implementation Patterns

5.2.1 4.2.1 Few-Shot Prompting for Text-to-Cypher

```
CYPHER_GENERATION_PROMPT = """
```

You are a Neo4j Cypher expert. Convert natural language to Cypher queries.

Schema:

- Nodes: (:Drug {name: string})
- Relationships: [:INTERACTS_WITH {severity: string}]
- Severity values: "major", "moderate", "minor"

Examples:

User: What drugs interact with Aspirin?

Cypher: MATCH (d:Drug {name: 'Aspirin'})-[r:INTERACTS_WITH]-(other:Drug)
RETURN other.name AS drug, r.severity AS severity

User: Does Warfarin interact with Ibuprofen?

Cypher: MATCH (d1:Drug {name: 'Warfarin'})-[r:INTERACTS_WITH]-(d2:Drug {name: 'Ibuprofen'})
RETURN d1.name, d2.name, r.severity

User: What are the major interactions with Metformin?

Cypher: MATCH (d:Drug {name: 'Metformin'})-[r:INTERACTS_WITH {severity: 'major'}]-(other:Drug)
RETURN other.name AS drug, r.severity AS severity

Now convert this query:

User: {user_query}

Cypher:

"""

5.2.2 4.2.2 Self-Healing Query Recovery

```
def execute_with_retry(self, natural_language_query: str, max_retries: int = 3):
    last_error = None

    for attempt in range(max_retries):
        try:
            # Generate Cypher from natural language
            cypher = self.generate_cypher(natural_language_query)

            # Execute against Neo4j
            result = self.graph.run(cypher)
            return result.data()

        except CypherSyntaxError as e:
```

```

last_error = str(e)

# Feed error back to LLM for correction
cypher = self.repair_query(
    original_query=natural_language_query,
    failed_cypher=cypher,
    error_message=last_error,
    schema=self.get_schema()
)

raise QueryFailedException(
    f"Failed after {max_retries} attempts. Last error: {last_error}"
)

```

5.2.3 4.2.3 Allergy Cross-Checking

```

DRUG_CLASS_MAPPING = {
    "Penicillin": ["Amoxicillin", "Ampicillin", "Piperacillin"],
    "Sulfa": ["Sulfamethoxazole", "Sulfasalazine"],
    "NSAIDs": ["Ibuprofen", "Naproxen", "Aspirin", "Celecoxib"],
    # ... additional mappings
}

def check_allergy_conflicts(patient: PatientProfile, medications: List[str]):
    alerts = []

    for allergy in patient.allergies:
        if allergy in DRUG_CLASS_MAPPING:
            # Check if any prescribed med is in the allergy class
            for med in medications:
                if med in DRUG_CLASS_MAPPING[allergy]:
                    alerts.append({
                        "type": "ALLERGY_ALERT",
                        "severity": "critical",
                        "allergy": allergy,
                        "medication": med,
                        "message": f"Patient allergic to {allergy}. "
                                   f"{med} is in the same drug class."
                    })

```

```
return alerts
```

5.3 Technology Stack

Layer	Technology	Version	Purpose
Database	Neo4j	5.x	Knowledge graph storage
LLM	Ollama + Llama 3.2	3B	Query generation, explanations
Orchestration	LangChain	0.1.x	Prompt management
Visualization	PyVis	0.3.x	Interactive graph networks
UI	Streamlit	1.28+	Web interface
Container	Docker	24.x	Service orchestration

6 Evaluation

6.1 Evaluation Framework

6.1.1 5.1.1 Test Case Categories

Category	Count	Description
Direct Interaction	20	Verify known drug pairs interact correctly
No Interaction	15	Verify system correctly reports no interaction
Single Drug Query	10	Retrieve all interactions for a single drug
Severity Filter	3	Filter interactions by severity level

6.1.2 5.1.2 Test Case Examples

Direct Interaction Test:

```
{
  "id": "DIRECT_001",
  "category": "direct_interaction",
  "question": "Does Mifepristone interact with Desoximetasone?",
  "drugs": ["Mifepristone", "Desoximetasone"],
  "expected_has_interaction": true,
  "expected_severity": "moderate"
}
```

No Interaction Test:

```
{
  "id": "NO_INT_001",
  "category": "no_interaction",
  "question": "Does Valrubicin interact with Physostigmine?",
  "drugs": ["Valrubicin", "Physostigmine"],
  "expected_has_interaction": false
}
```

6.2 Evaluation Results

6.2.1 5.2.1 Overall Performance

Metric	Value
Total Test Cases	48
Passed	48
Failed	0
Overall Accuracy	100.0%

6.2.2 5.2.2 Accuracy by Category

Category	Tests	Passed	Accuracy
Direct Interaction	20	20	100.0%
No Interaction	15	15	100.0%
Single Drug Query	10	10	100.0%
Severity Filter	3	3	100.0%

6.2.3 5.2.3 Response Time Performance

Metric	Value
Average Response Time	2,009 ms
Minimum Response Time	1,387 ms
Maximum Response Time	6,549 ms
Evaluation Duration	96.4 seconds

6.2.4 5.2.4 Text-to-Cypher Performance

Metric	Value
First Attempt Success	47/48 (97.9%)
Required Retry	1
Self-Healing Rate	100.0%

The single query that required retry was successfully recovered through the self-healing mechanism, demonstrating the effectiveness of the error feedback loop.

6.2.5 5.2.5 Severity Classification

Metric	Value
Severity Tests	23
Correct Classifications	20
Severity Accuracy	87.0%

6.2.6 5.2.6 Patient Risk Scoring Validation

Three synthetic patients were created to validate risk score ordering:

Patient ID	Age	Kidney Function	Medications	Base Score	Adjusted Score	Risk Level
TEST_LOW_85		Normal	2	10.0	10.0	Low
TEST_MED_70		Mild Impairment	4	3.3	16.8	Low
TEST_HIGH_90		Severe Impairment	6	13.3	92.7	Critical

Validation Results:

- ☐ Risk ordering correct (Low < Low < Critical)
- ☐ High-risk patient correctly identified
- ☐ Multipliers applied appropriately (90-year-old with severe kidney impairment: ~7x multiplier)

7 User Interface

7.1 Streamlit Application

The application provides six specialized tabs:

7.1.1 6.1.1 Patient Analysis Tab

Primary interface for patient-centered medication review:

1. Search existing patients by name or ID
2. Filter patients by risk level
3. View patient demographics and risk factors
4. Run personalized polypharmacy analysis
5. View interactive interaction graph
6. Read LLM-generated plain-language summary

7.1.2 6.1.2 Quick Check Tab

Rapid multi-drug interaction check without patient context:

- Enter 2-10 medications
- View all pairwise interactions
- See severity distribution
- Generate basic risk score

7.1.3 6.1.3 Natural Language Tab

Free-form query interface:

- “What drugs interact with Warfarin?”
- “Show me major interactions for Metformin”
- “Does Aspirin interact with Lisinopril?”

7.1.4 6.1.4 Search Drug Tab

Single drug exploration:

- Search by drug name
- View all known interactions
- Filter by severity
- See interaction network graph

7.1.5 6.1.5 Two Drugs Tab

Direct pairwise interaction check:

- Select two drugs from dropdowns
- Immediate interaction result
- Severity and description display

7.1.6 6.1.6 Browse Tab

Database exploration:

- View all drugs in knowledge graph
- Statistics and distribution charts
- Random sampling for exploration

8 Deployment

8.1 Local Development Setup

8.1.1 Prerequisites

- Python 3.11+
- Docker and Docker Compose
- 8GB+ RAM (for Ollama)
- macOS with Metal (recommended) or NVIDIA GPU

8.1.2 Installation Steps

```
# Clone repository
git clone https://github.com/pranavkharat/medgraphrag.git
cd medgraphrag

# Create virtual environment
python -m venv venv
source venv/bin/activate

# Install dependencies
pip install -r requirements.txt

# Start Neo4j
docker-compose up -d

# Install Ollama (macOS)
brew install ollama

# Pull model
ollama pull llama3.2:3b

# Start Ollama server
ollama serve &

# Load data into Neo4j
python -m src.ingestion.csv_parser

# Generate synthetic patients
```

```
python -m src.data.synthetic_patients
```

```
# Run application
streamlit run app/main.py
```

8.1.3 Service URLs

Service	URL
Streamlit App	http://localhost:8501
Neo4j Browser	http://localhost:7474
Ollama API	http://localhost:11434

8.2 Running Evaluation

```
# Ensure services are running
docker-compose up -d
ollama serve &

# Run full evaluation suite
python -m src.evaluation.run_evaluation

# Output files generated:
# - data/evaluation/metrics.json
# - data/evaluation/evaluation_report.txt
```

9 Future Work

9.1 Planned Enhancements

9.1.1 8.1.1 Hybrid RAG Integration

Add vector embeddings alongside graph structure for semantic drug similarity:

- Find drugs with similar mechanisms of action
- Identify potential substitutes when interactions are detected
- Enable fuzzy drug name matching

9.1.2 8.1.2 Fine-Tuned Text-to-Cypher Model

Train a specialized model for Cypher generation:

- Reduce first-attempt failure rate
- Improve response latency
- Enable deployment without large LLM

9.1.3 8.1.3 FHIR Integration

Connect to Electronic Health Record systems:

- Pull patient data directly from EHR
- Push alerts to clinical workflow
- Maintain audit trail for compliance

9.1.4 8.1.4 Multi-Language Support

Extend LLM explanations to support:

- Spanish
- Mandarin
- Hindi
- Other high-demand languages

9.2 Scalability Considerations

Component	Current	Production Path
Patient Data	JSON file	PostgreSQL with encryption
Drug Lookups	Direct Neo4j	Redis cache layer

Component	Current	Production Path
LLM Calls	Synchronous	Async batch processing
Deployment	Local Docker	Kubernetes cluster

10 Appendix

10.1 A. Complete Evaluation Metrics

```
{
  "total_tests": 48,
  "passed_tests": 48,
  "failed_tests": 0,
  "overall_accuracy": 100.0,
  "category_results": {
    "direct_interaction": {"total": 20, "passed": 20, "accuracy": 100.0},
    "no_interaction": {"total": 15, "passed": 15, "accuracy": 100.0},
    "single_drug_query": {"total": 10, "passed": 10, "accuracy": 100.0},
    "severity_filter": {"total": 3, "passed": 3, "accuracy": 100.0}
  },
  "avg_response_time_ms": 2008.74,
  "first_attempt_success": 47,
  "required_retry": 1,
  "self_healing_rate": 100.0,
  "severity_accuracy": 86.96,
  "patient_risk_scoring": {
    "patients_tested": 3,
    "risk_ordering_correct": true,
    "high_risk_identified": true
  }
}
```

10.2 B. Sample Cypher Queries

Find all interactions for a drug:

```
MATCH (d:Drug {name: 'Warfarin'})-[r:INTERACTS_WITH]-(other:Drug)
RETURN other.name AS drug, r.severity AS severity
ORDER BY r.severity
```

Check specific drug pair:

```
MATCH (d1:Drug {name: 'Aspirin'})-[r:INTERACTS_WITH]-(d2:Drug {name: 'Ibuprofen'})
RETURN d1.name, d2.name, r.severity, r.description
```

Get major interactions only:

```
MATCH (d:Drug {name: 'Metformin'})-[r:INTERACTS_WITH {severity: 'major'}]-(other:Drug)
RETURN other.name AS drug
```

Count interactions by severity:

```
MATCH (d:Drug {name: 'Lisinopril'})-[r:INTERACTS_WITH]-(other:Drug)
RETURN r.severity AS severity, count(*) AS count
ORDER BY count DESC
```

10.3 C. Environment Variables

```
# .env file
NEO4J_URI=bolt://localhost:7687
NEO4J_USER=neo4j
NEO4J_PASSWORD=password
OLLAMA_BASE_URL=http://localhost:11434
OLLAMA_MODEL=llama3.2:3b
```

10.4 D. References

1. DrugBank Database - <https://go.drugbank.com/>
 2. Neo4j Graph Database - <https://neo4j.com/>
 3. Ollama - <https://ollama.ai/>
 4. LangChain Documentation - <https://python.langchain.com/>
 5. Streamlit - <https://streamlit.io/>
-

Document Version: 1.0

Last Updated: December 2025 **Author:** Pranav Kharat

Institution: Northeastern University

Repository: <https://github.com/pranavkharat/medgraphrag>