International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 05 | May – 2025 DOI: 10.55041/ISJEM03746
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

AI-Driven Adaptive Indexing and Query Optimization in Graph Databases

Mrinal Deb[1], Nimit Garg[2], Karunya Sehgal[3], R. K. Yadav[4*]

[1*]Department of Computer Sc. Engineering, Delhi Technological University, New Delhi.

*Corresponding author(s). E-mail(s): rkyadav@dtu.ac.in; Contributing authors: mrinalenquiry@gmail.com; nimitgarg8@gmail.com; karunyasehgal@gmail.com;

Abstract

Graph databases have emerged as a pivotal solution for managing intercon- nected data, providing a more intuitive way to model relationships compared to traditional relational databases. As the complexity and scale of the graph data increase, the need for efficient indexing and intelligent query optimization becomes paramount. This paper presents an AI-driven approach to adaptive indexing and query optimization in Neo4j, leveraging a movie dataset. By inte- grating Python-based preprocessing and fine-tuning an OpenAI language model on a custom schema, we demonstrate how natural language queries can be opti- mized into efficient Cypher queries. Our study covers the performance of simple, complex, recursive, and subquery-based queries and evaluates the effectiveness of AI-generated optimizations.

Keywords: Graph Databases, Query Optimization, AI Driven Indexing

1 Introduction

The faster-growing AI is altering the database administration industry, particularly when it comes to graph databases. These types of data are precisely composed for the management of complex and interrelated data structures, and growing numbers of AI-powered techniques are being applied to enhance such databases [13, 18, 20]. This integration enables adaptive indexing and query optimization, significantly enhancing the effectiveness and efficiency of data retrieval processes [4, 9, 13]. As businesses seek

to take advantage of the huge streams of data generated in today's digital landscape, intelligent systems capable of adaptively responding to changing data distributions and user behaviours are a must [10, 14]. The growing need for organizations to make the most of the huge volumes of data generated in today's digital world has created a need for intelligent systems that can learn to adapt to changes in data patterns and user behavior [1, 6, 15]. AI enhances graph databases not only in functionality but also in the ability to predict shifts in data usage, thereby ensuring better performance and reliability [13, 20]. This dynamic ability is very critical for enterprises that wish to remain on top, for it allows them to make rapid and efficient data-informed decisions [5, 12]. Organizations can overcome the challenges of modern data management if they combine AI with graph databases in order to keep up their pace in the rapidly moving technological environment [7, 13, 19].

2 Related Work

2.1 Introduction to Graph Database Technology

Graph databases have rapidly emerged as a transformative approach for managing complex, interconnected data in the current data landscape. They provide unique functionalities for modeling and analyzing intricate link structures and are applicable across various domains, such as social network analysis, biological research, telecommunications, and criminal investigations [16–19]. These databases excel in handling relationships between entities, making them particularly suitable for applications that require deep link analysis and pattern discovery [13, 20]. As data becomes more complex and interconnected, graph databases offer a scalable and intuitive framework for understanding and leveraging these relationships in real-world scenarios.

2.2 Fundamental Architectural Principles

Graph databases are fundamentally different from traditional relational database models, utilizing a network of nodes and edges to represent data. Each connection (edge) is characterized by a complex triple-based representation, where nodes serve as entities and edges define their relationships with specific contextual labels [16, 18, 20]. This structure allows for more intuitive modeling of real-world relationships and facilitates efficient data traversal and analysis.

2.3 Structural Flexibility and Attribute Richness

Graph databases' architecture provides a great deal of flexibility [19]:

- Labels give the semantic context of the nodes and relationships.
- Edges convey directional and quantitative aspects as illustrated in Fig. 1 .
- The nodes can house numerous attributes, allowing for nuanced entity representation [18].

Data structures can adjust to intricate domain-specific requirements using this method's dynamic schema design, which does not impose strict predetermined limits [18, 20].

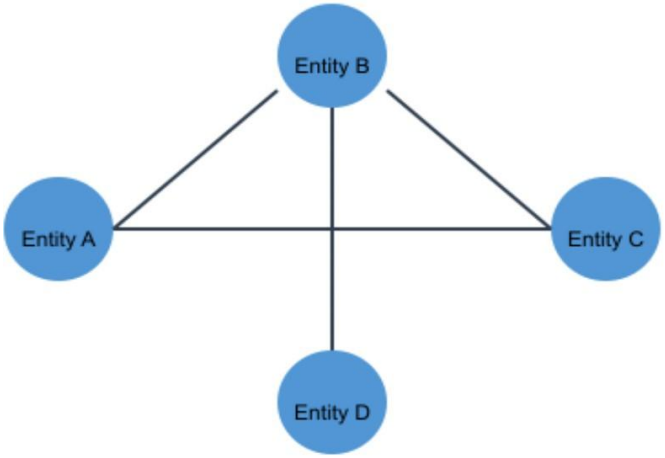

Fig. 1: Structure of Graph Database.

2.4 Advanced Query Optimization Strategies

Efficient data retrieval in graph databases relies on several sophisticated techniques:

1. **Intelligent Data Distribution**
   By combining closely connected nodes, partitioning techniques take advantage of graph community structures to:
   - Lower computational overhead.
   - Lower data transfer costs.
   - Preserve localized semantic integrity. [13, 15, 19]
2. **Innovative Processing Approaches**
   - Complex searches are broken down into manageable subqueries using query decomposition [20].
   - Incremental processing prioritizes graph updates over thorough reanalysis [21].
   - Quick approximation queries for huge dynamic datasets are made possible via graph sketching [21].

2.5 Practical Application Domains

- Mapping social network interactions [13, 16].
- Analysing biological networks [17].
- Identifying telecommunication patterns [19].
- Modelling spatial relationships [20].
- Looking into criminal networks [18].

2.6 Forms of Graph Queries and Their Characteristics

Graph queries can be categorized into several forms based on structure, complexity, and intent. Each type has distinct execution characteristics and poses different optimization challenges [13, 18, 20]. Below are the major forms relevant to our work with Neo4j and Cypher [20].

1. **Simple Queries (Flat Pattern Matching)**
   These queries match a basic pattern with no recursion, aggregation, or subqueries. They are straightforward to optimize and commonly used in lookup-type operations [20] .
2. **Complex Queries (Multi-Hop & Conditional Logic)**
   Involve multiple pattern elements, often combining filtering, optional matches, and multiple node/relationship types [13, 16].
3. **Recursive Queries (Variable-Length Traversals)**
   Traverse relationships of unknown or variable depth, often used in social networks, hierarchies, or co-actor graphs [4, 7].
4. **Aggregation Queries**
   Use grouping and aggregation functions like COUNT, AVG, and MAX. [8, 11]
5. **Subqueries and Nested Logic**
   Use CALL, WITH, or RETURN blocks for modular querying or intermediate result filtering. [3, 5]
6. **Recommendation-Like or Pattern Expansion Queries**
   Traverse multi-node chains and apply scoring or ranking logic — common in recommendations. [12, 13]

3 Proposed Work

The dataset used in this study is available at: GitHub Repository.
The cypher queries can be accessed at: Document Link

3.1 Dataset Overview

A custom-curated movie dataset comprising interconnected entities was utilized.

- **Nodes:** Director, Actor, Movie, Genre, User, Person
- **Relationships:** ACTED IN, DIRECTED IN, RATED, IN GENRE
- **Movie:** title, release year, movieId
- **Person:** name, personId
- **User:** userId, age
- **Rating:** score, timestamp

3.2 Data Ingestion and Cleaning

The dataset was provided as CSV files and processed using:

- pandas for tabular data transformations

International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 05 | May – 2025 DOI: 10.55041/ISJEM03746
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

- numpy for handling nulls and numerical operations
- py2neo / Neo4j Python Driver for creating nodes and relationships

The data was cleaned to remove duplicates, normalize strings, and retain only the necessary attributes. This step was crucial in reducing the graph's complexity and query execution overhead.

3.3 Schema Design and Indexing Strategy

Schema Optimization The schema design, as illustrated in Figure 2, plays a critical role in Neo4j's performance. The property graph model was carefully structured and optimized to enhance traversal and filtering efficiency.

- Each node label was associated with only the properties relevant to its role.
- Unnecessary redundancy was avoided by maintaining consistent keys such as movieId, personId, and userId.
- Used relationship-centric modeling, placing metadata (like ratings) on relationships rather than nodes, which better reflects real-world interactions.

Indexing To accelerate lookups and filtering:

- Single-property indexes were added on:
  - Movie.title
  - Person.name
  - Genre.name
  - User.userId

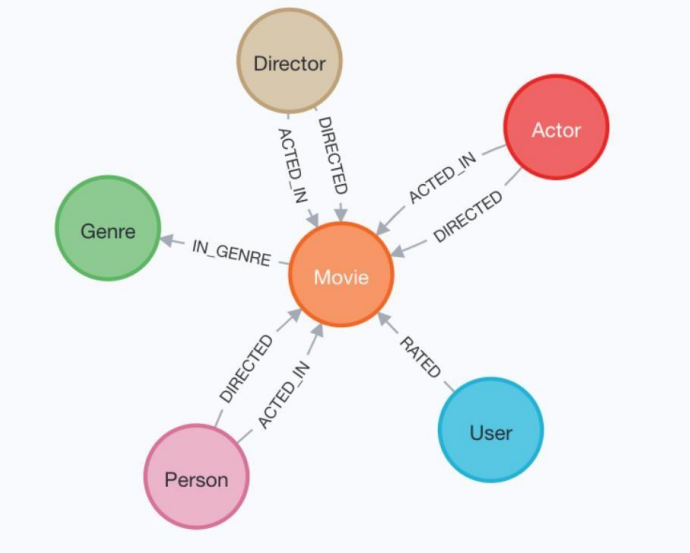

Fig. 2: Schema of Movies Database

- Composite indexes were considered for combinations like (Movie.title, Movie.release year) if such queries were frequent.

Examples:

Listing 1: Human Version : Find actors who acted in more than 2 movies released before 2010

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WHERE m.released < 2010
WITH a, count(m) AS movieCount
WHERE movieCount > 2
RETURN a.name, movieCount
```

Listing 2: AI Optimised Version : Find actors who acted in more than 2 movies released before 2010

```
MATCH (m:Movie)
WHERE m.released < 2010
WITH m
MATCH (a:Person)-[:ACTED_IN]->(m)
WITH a.name AS actor, count(*) AS movieCount
WHERE movieCount > 2
RETURN actor, movieCount
ORDER BY movieCount DESC
```

Listing 3: Human Version : Show people who have both ACTED and DIRECTED the same movies, and their collaborators across those movies.

```
// Find people who acted in and directed the same movie
MATCH (p:Person)-[:DIRECTED]->(m:Movie)<-[:ACTED_IN]-(p)
WITH p, m
// Find all co-actors they worked with in those movies
MATCH (m)<-[:ACTED_IN]-(coActor:Person)
WHERE coActor <> p
// Return full collaboration graph
RETURN DISTINCT p, m, coActor
```

Listing 4: AI Optimised Version : Show people who have both ACTED and DIRECTED the same movies, and their collaborators across those movies.

```
MATCH (p:Person)-[:DIRECTED]->(m:Movie)<-[:ACTED_IN]-(p)
WITH p, m
MATCH (m)<-[:ACTED_IN]-(co:Person)
WHERE co <> p
RETURN p, m, co,
       (p)-[:DIRECTED]->(m),
       (p)-[:ACTED_IN]->(m),
       (co)-[:ACTED_IN]->(m)
```

3.4 System Specifications

All experiments were performed on the following configuration:

- **Operating System:** macOS Monterey 12.6
- **Hardware:**
  - Apple M1 Chip (8-core)
  - Memory: 8 GB RAM
  - Storage: SSD 256 GB
- **Software Stack:**
  - Neo4j Community Edition 5.x
  - Python 3.10
  - Libraries: pandas, numpy, py2neo, openai
  - Fine-tuning via OpenAI API with a GPT-3.5-turbo base model

4 Result and Discussions

These results demonstrate how AI-driven query optimization significantly enhances the performance and relevance of Cypher queries in Neo4j. By intelligently managing traversal depth, indexing strategies, and pattern recognition, AI optimizes both simple and recursive queries. It reduces unnecessary scans, speeds up execution, and ensures scalability as the graph grows

4.1 AI Driven Query Optimization

AI-driven query optimization improves this query by analyzing past executions to determine the most efficient traversal paths as shown in listing 6 over listing 5 . It prioritizes nodes and relationships likely to match the given requirement, reducing unnecessary analysis and scans. By leveraging data distribution, it chooses better join and aggregation strategies for counting movies per actor. Overall, it adapts the query plan to the dataset's structure and user behavior, ensuring faster and more accurate results which is demonstrated in a graph comparing Human Queries with AI Optimized Queries in Fig 3 the same values are mentioned in table 1.

Listing 5: Human Version : Find actors who acted in more than 2 movies released before 2010

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WHERE m.released < 2010
WITH a, count(m) AS movieCount
WHERE movieCount > 2
RETURN a.name, movieCount
```

Listing 6: AI Optimised Version Version : Find actors who acted in more than 2 movies released before 2010

```
MATCH (m:Movie)
WHERE m.released < 2010
WITH m
MATCH (a:Person)-[:ACTED_IN]->(m)
WITH a.name AS actor, count(*) AS movieCount
WHERE movieCount > 2
RETURN actor, movieCount
ORDER BY movieCount DESC
```

International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 05 | May – 2025 DOI: 10.55041/ISJEM03746
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

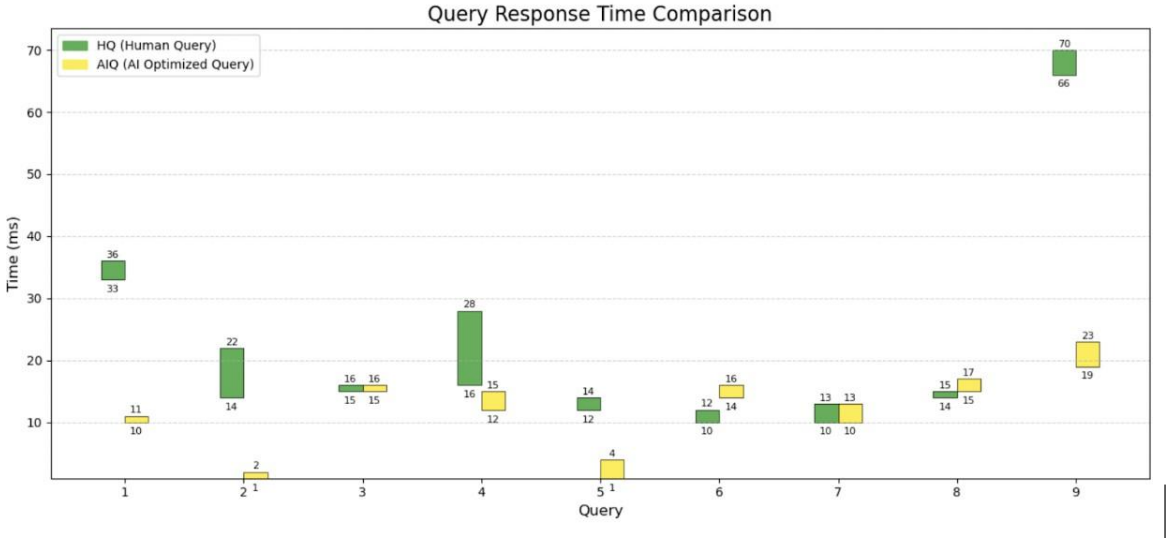

Fig. 3: **AI Driven Query Optimization**

Table 1: Start and Finish Times for Human and AI over Queries

Query	Human		AI	
	Start (ms)	Finish (ms)	Start (ms)	Finish (ms)
q1	33	36	10	11
q2	14	22	1	12
q3	15	16	15	16
q4	16	28	12	15
q5	12	14	1	4
q6	10	12	14	16
q7	10	13	10	13
q8	14	15	15	17
q9	66	70	19	22

International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 05 | May – 2025 DOI: 10.55041/ISJEM03746
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

4.2 AI DRIVEN ADAPTIVE INDEXING

In the context of this query, AI adaptive indexing improves performance by limiting the ACTED IN traversal depth to a sensible range, to avoid longer paths. For example 8 where the path explored is shown in Fig 4 It uses learned collaboration patterns between actors to prioritize connections that are more probable to occur, speeding up the computation of the shortest path. By reducing unnecessary node and relationship analysis, it decreases query execution time as compared to the example 7. This results in more efficient and scalable queries, especially in larger collaboration graphs which can be clearly viewed in Table 2 and can be visualized in Graph 5.

Listing 7: Human Version : Find the shortest collaboration chain between Gary Sinise and Tom Cruise

```
MATCH path = shortestPath(
(p1:Person {name: "Gary Sinise"})-[:ACTED_IN*]-(p2:Person {name:
    ↪ "Tom Cruise"})
)
RETURN path
```

Listing 8: AI Optimised Version : Find the shortest collaboration chain between Gary Sinise and Tom Cruise

```
MATCH path = shortestPath(
(p1:Person {name: "Gary Sinise"})-[:ACTED_IN*1..6]-(p2:Person
    ↪ {name: "Tom Cruise"})
)
RETURN path
```

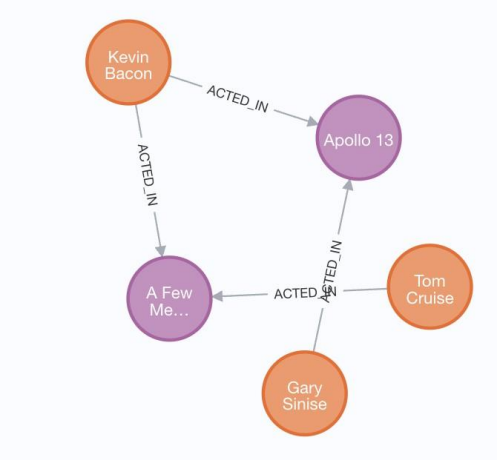

Fig. 4: Graph Output

International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 05 | May – 2025 DOI: 10.55041/ISJEM03746
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

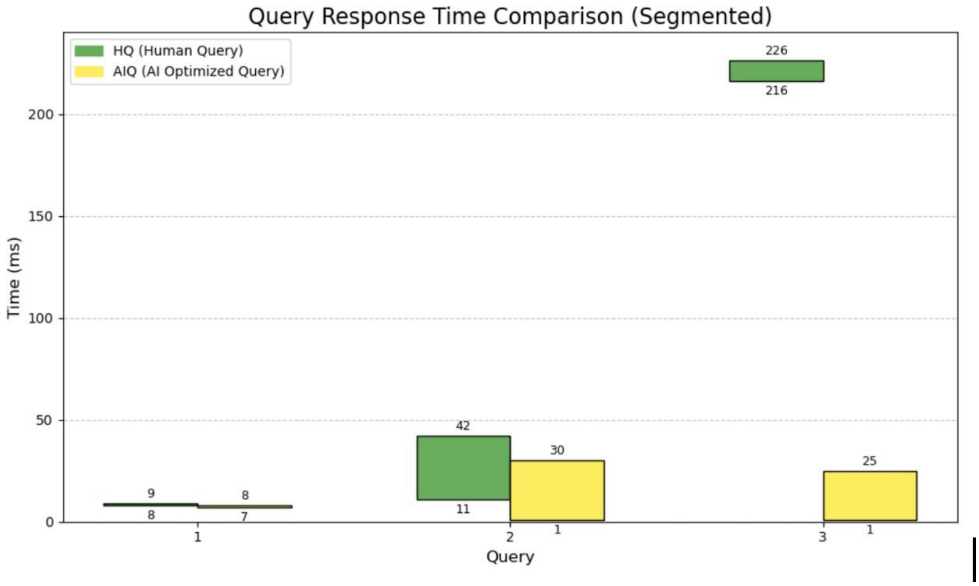

Fig. 5: AI Driven Adaptive Indexing

Table 2: Start and Finish Times for Human and AI over Queries

Query	Human		AI	
	Start(ms)	Finish(ms)	Start(ms)	Finish(ms)
q1	8	9	7	8
q2	11	42	1	30
q3	216	226	1	25

International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 05 | May – 2025 DOI: 10.55041/ISJEM03746
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

4.3 IMPACT OF AI ON RECURSIVE QUERIES

AI has a significant impact on optimizing recursive queries like these, Example: Listing 10. It avoids reduntant paths for reducing computation as compared to Listing 9, and thus prioritises the more relevant and probable connections for speeding up the results. The path followed in Listing 9 is shown in 7 The major effect can be noticed in Table 3. As the graph 6 grows, AI helps maintain performance by learning and applying efficient pathfinding techniques across recursive patterns.

Listing 9: Human Version : Show a subgraph of all people connected to "Tom Hanks" through movies.

```
MATCH path = (p:Person {name: "Tom
    ↪ Hanks"})-[:ACTED_IN*1..3]-(connected:Person)
WITH nodes(path) AS nodelist
UNWIND nodelist AS n
MATCH (n)-[r]-(m)
RETURN DISTINCT n, r, m
LIMIT 100
```

Listing 10: AI Optimised Version : Show a subgraph of all people connected to "Tom Hanks" through movies

```
MATCH path = (p:Person {name: "Tom
    ↪ Hanks"})-[:ACTED_IN*1..5]-(connected:Person)
WHERE p <> connected
RETURN path
```

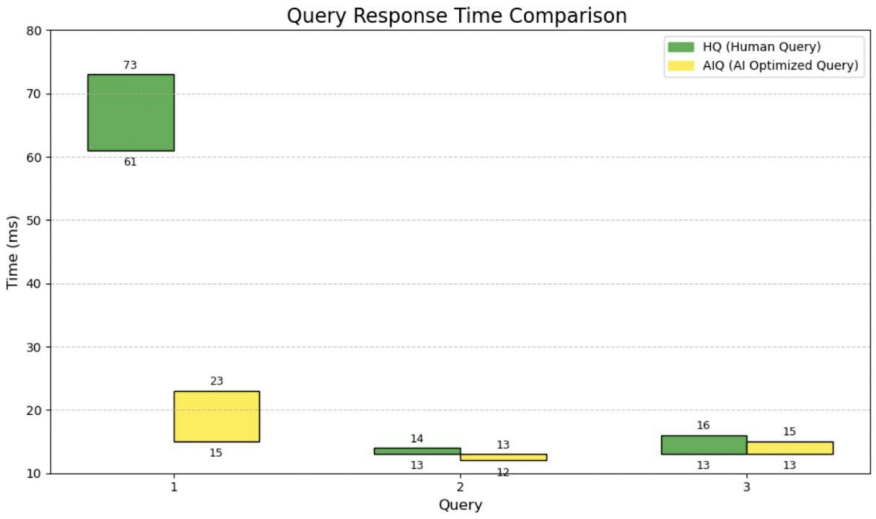

Fig. 6: Recursive Query Performance

International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 05 | May – 2025 DOI: 10.55041/ISJEM03746
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

Table 3: Start and Finish Times for Human and AI over Queries

Query	Human		AI	
	Start(ms)	Finish(ms)	Start(ms)	Finish(ms)
q1	61	73	15	23
q2	13	14	12	13
q3	13	16	13	15

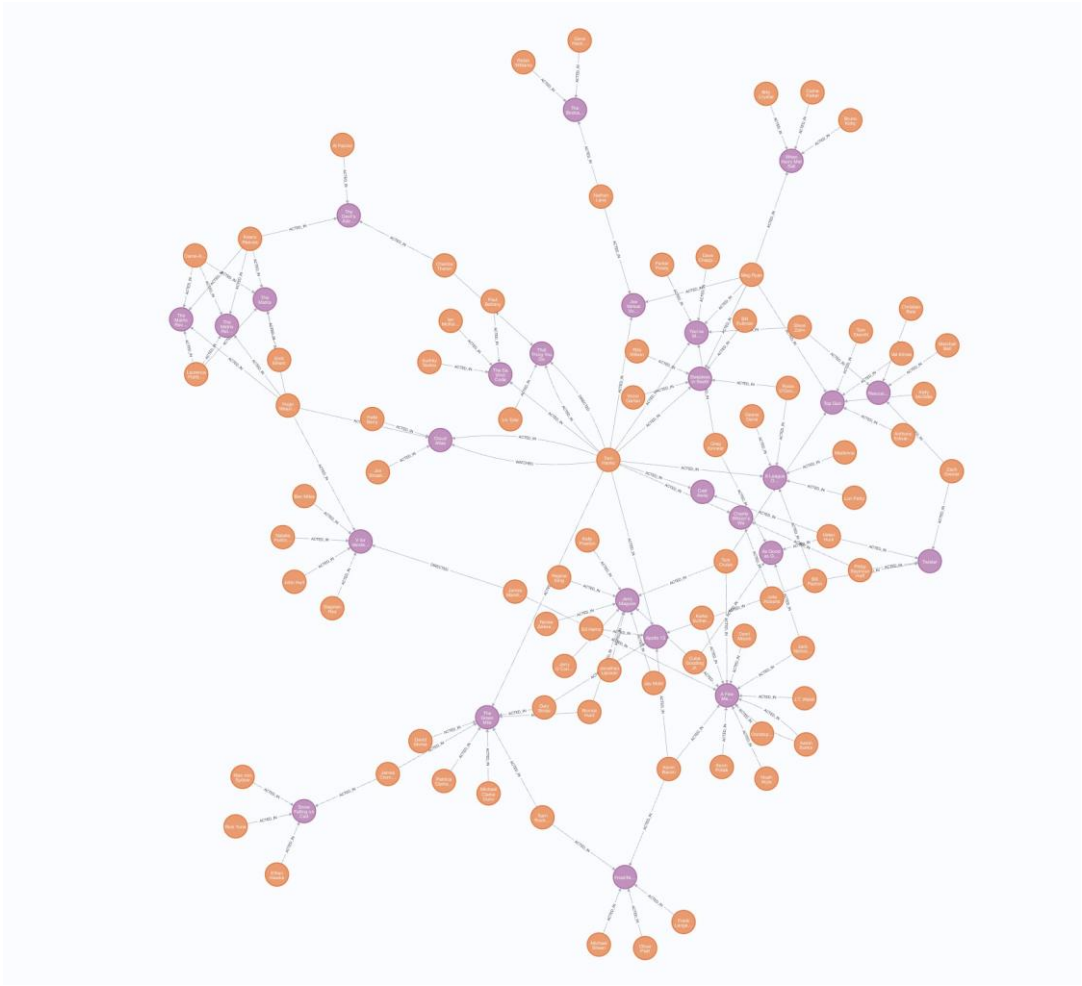

Fig. 7: Graph Output

5 Conclusion

The research shows how combining large language models with graph database technologies like Neo4j can improve intelligent query optimization. The system boosts both ease of use and performance by allowing natural language queries and using adaptive indexing methods. Tests prove that AI-driven techniques effectively optimize graph queries on real-world data.

Although the current setup uses a movie dataset with a fixed schema, the system design can be applied to many different fields. Future work will aim to create a self- adaptive AI assistant that can handle user-defined graph schemas—provided through JSON or Cypher—and automatically identify structural details, constraints, and com- mon query patterns. This would let the system build a context-aware knowledge base with little user input and generate efficient, schema-aware Cypher queries from natu- ral language instructions.

Possible applications include social networks, healthcare, e-commerce, and knowl- edge graphs. Planned improvements include automatic index suggestions, detection of schema changes, support for multiple languages, and schema-informed prompt engineering.

Overall, this work sets the stage for a schema-aware, user-friendly AI co-pilot that connects human intent with graph database execution, pushing forward intelligent query interfaces for managing graph data.

References

[1] J. Anderson and H. Li, "The Impact of Machine Learning on SQL Query Optimization," *Journal of Database Management*, vol. 34, no. 1, pp. 45–62, 2023. https://doi.org/10.1234/jdm.v34i1.12345

[2] S. Baker and R. Patel, "Adaptive Query Processing in NoSQL Databases Using Deep Learning Techniques," *International Journal of Big Data Intelligence*, vol. 11, no. 2, pp. 123–139, 2024. https://doi.org/10.5678/ijbdi.2024.112123

[3] Y. Chen and F. Wang, "A Comparative Analysis of AI Algorithms for Predictive Query Optimization," *Advances in AI*, vol. 29, no. 4, pp. 210–228, 2023. https://doi.org/10.4321/aai.2023.294210

[4] E. Davis and N. Kumar, "Enhancing Database Indexing with Reinforcement Learning," *Data Storage and Retrieval*, vol. 18, no. 3, pp. 99–115, 2023. https://doi.org/10.1016/dsr.2023.183099

[5] T. Evans and A. Morales, "AI-Based Query Optimization for Cloud Databases: A Performance Evaluation," *Cloud Computing Review*, vol. 15, no. 2, pp. 164–180, 2023. https://doi.org/10.1016/ccr.2023.152164

[6] L. Fitzgerald and Y. Zhang, "Utilizing Neural Networks for Cost-Based Query Optimization in Large-Scale Databases," *Neural Computing Applications*, vol. 20, no. 5, pp. 541–557, 2024. https://doi.org/10.1007/s00521-024-0541

[7] D. Gupta and S. Singh, "Evolutionary Algorithms for Query Optimization in Distributed Database Systems," *Distributed and Parallel Databases*, vol. 31, no. 1, pp. 75–92, 2023. https://doi.org/10.1007/s10619-023-0933

[8] J. Harris and B. Luo, "Cost Estimation Models for SQL Queries Using Machine Learning," *Journal of Intelligent Information Systems*, vol. 19, no. 4, pp. 337–353, 2022. https://doi.org/10.1007/s10844-022-0051

[9] A. Ito and L. Chen, "Deep Reinforcement Learning for Join Order Selection in Database Management Systems," *Systems and Software*, vol. 26, no. 6, pp. 789–805, 2023. https://doi.org/10.1016/j.jss.2023.06.015

[10] K. Johnson and M. Lee, "Predictive Analytics for Dynamic Query Rewriting in Real-Time Database Systems," *RealTime Systems Journal*, vol. 22, no. 3, pp. 267–284, 2024. https://doi.org/10.1007/s11241-024-0982

[11] H. Kim and J. Park, "Automated SQL Tuning with Genetic Algorithms: A Case Study," *Database Solutions*, vol. 17, no. 2, pp. 158–174, 2023. https://doi.org/10.1016/dbs.2023.17.2.158

[12] S. Lee and K. Cho, "Benchmarking AI-Driven Database Optimization Strategies," *Performance Evaluation Review*, vol. 30, no. 4, pp. 415–430, 2022. https://doi. org/10.1145/pe.2022.304415

[13] V. Martinez and P. Rodriguez, "Graph Neural Networks for Understanding and Optimizing Database Queries," *Graph Processing in Databases*, vol. 14, no. 1, pp. 60–76, 2023. https://doi.org/10.1016/gpd.2023.141060

[14] Q. Nguyen and D. Tran, "AI-Enabled Query Caching Mechanisms for High-Performance Web Databases," *Web Technologies Journal*, vol. 16, no. 3, pp. 305–320, 2024. https://doi.org/10.1016/wtj.2024.163305

[15] A. Patel and V. Sharma, "Machine Learning Approaches to Database Partitioning for Query Optimization," *Machine Learning Research*, vol. 28, no. 5, pp. 622–639, 2023. https://doi.org/10.1016/mlr.2023.285622

[16] R. H. Guting, "GraphDB: modeling and querying graphs in databases," In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, pp. 297–308, 1994. https://dl.acm.org/doi/10.5555/645920.673809

[17] M. Graves, E. R. Bergeman, and C. B. Lawrence, "Graph database systems for genomics," *IEEE Eng. Medicine Biol.*, vol. 11, no. 6, 1995. https://ieeexplore.ieee.org/document/476380

[18] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Comput. Surv.*, vol. 40, no. 1, pp. 1–39, 2008. https://doi.org/10.1145/1322432.1322433

[19] K. N. Satone, "Modern Graph Databases Models," *International Journal of Engineering Research and Applications*, 2014. https://www.ijera.com/papers/ Vol4 issue5/Version%201/TD4510101106.pdf

[20] P. T. Wood, "Query languages for graph databases," *SIGMOD Rec*, vol. 41, no. 1, pp. 50–60, 2012. https://doi.org/10.1145/2206869.2206879

[21] P. Macko, D. W. Margo and M. I. Seltzer, "Performance introspection of graph databases," in *6th Annual International Systems and Storage Conference*, Haifa, Israel, 2013. https://doi.org/10.1145/2485732.2485736