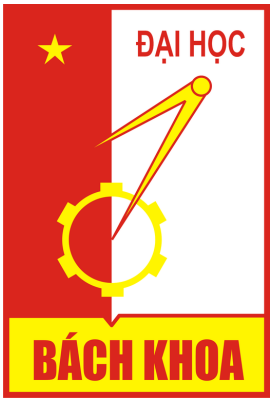
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY



Semantic Web

GraphRAG Neo4j Chatbot for London Transport Information

Vu Tuan Kiet - 20242434M
kiet.vt242434M@sis.hust.edu.vn

Nguyen Ngoc Tu - 20251189M
tu.nn20251189M@sis.hust.edu.vn

Course: IT6390E

Supervisor: Dr. Do Ba Lam
School: Information and Communications Technology

HANOI, 08/2025

ABSTRACT

This report presents a comprehensive analysis of a Graph-based Retrieval Augmented Generation (GraphRAG) system designed for London's transportation network. The system integrates Transport for London (TfL) API data with Neo4j graph database technology and Large Language Models (LLMs) to enable natural language querying of complex transportation information. We examine the system architecture, evaluate its adherence to semantic web principles, and assess its effectiveness in addressing urban transportation data complexity. Our analysis reveals that the GraphRAG approach successfully transforms hierarchical JSON data into semantically rich graph representations while providing robust natural language interfaces for non-technical users.

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

1.1 Motivation

Urban transportation systems generate vast amounts of interconnected data encompassing network topology, service schedules, real-time incidents, and accessibility information. Traditional relational database approaches struggle to capture the complex relationships inherent in transportation networks, leading to data silos and inefficient information retrieval [1]. The emergence of knowledge graphs and semantic web technologies offers promising solutions for modeling these complex relationships while maintaining query flexibility and semantic expressiveness.
Graph-based Retrieval Augmented Generation (GraphRAG) represents a novel approach that combines the structural advantages of knowledge graphs with the natural language capabilities of Large Language Models (LLMs). This paradigm enables users to query complex graph structures using natural language, potentially democratizing access to transportation information systems.

1.2 Objectives

This study examines a GraphRAG implementation for London's transportation network with the following objectives:

1. Analyze the system's graph data modeling approach and its alignment with semantic web principles
2. Evaluate the natural language query translation mechanism and its accuracy
3. Assess the system's error handling and robustness in real-world scenarios
4. Identify strengths, limitations, and areas for improvement in the current implementation

1.3 Scope and Limitations

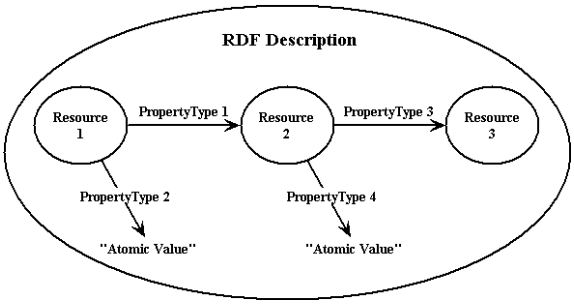
The project focuses on implementation that integrates TfL API data, Neo4j graph database, and OpenAI's GPT-4 model. It is limited to London's transportation network and does not encompass multi-city or international transportation systems. Also, due to problem of cost, the project can not execute realtime data to represent.

CHAPTER 2. RELATED WORK

2.1 Semantic web knowledge graph

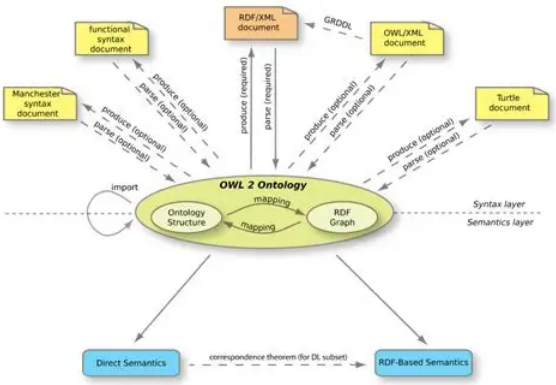
2.1.1 Resource Description Framework (RDF)

The Resource Description Framework (RDF) is a method for describing and exchanging graph data, originally designed by the World Wide Web Consortium (W3C) as a data model for metadata. RDF represents data as a directed graph composed of triple statements, where each statement consists of a subject, a predicate, and an object. These components can be identified by Uniform Resource Identifiers (URIs), and objects can also be literal values. The framework enables semantic interoperability across different systems and supports reasoning through its formal semantics. RDF's standardized approach to data representation has made it the foundation for linked data initiatives and semantic web applications worldwide.



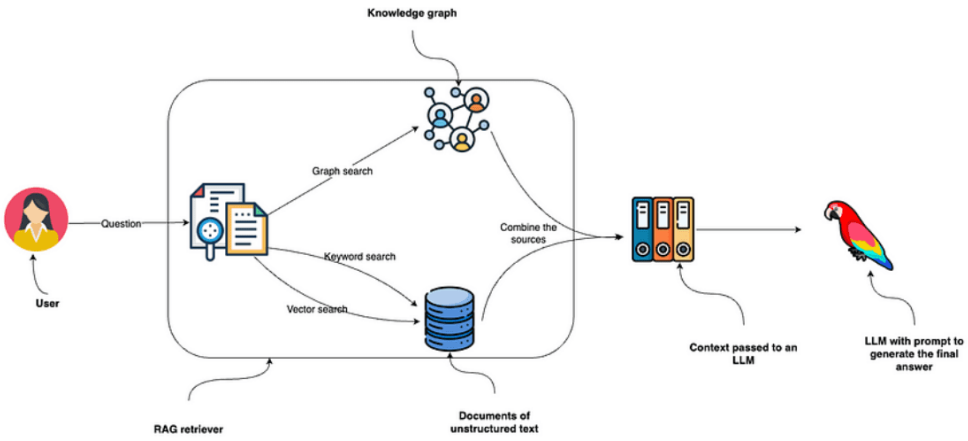
2.1.2 Web Ontology Language

The Web Ontology Language (OWL) is a semantic web language designed to represent rich and complex knowledge about things, groups of things, and the relationships between them. It is built upon the Resource Description Framework (RDF) and is widely used for creating ontologies in domains like healthcare, e-commerce, and artificial intelligence. OWL provides three sublanguages of increasing expressiveness: OWL Lite, OWL DL, and OWL Full, each offering different levels of computational complexity and reasoning capabilities. The language supports formal reasoning through description logic, enabling automated inference of implicit knowledge from explicitly stated facts.



2.2 Graph Retrieval-Augmented Generation (GraphRAG)

GraphRAG represents an evolution of traditional RAG systems by incorporating graph structures to maintain contextual relationships between entities. Unlike vector-based RAG systems, GraphRAG preserves the semantic relationships inherent in connected data, making it particularly suitable for transport networks where stations, lines, and routes form natural graph structures. This approach enables more sophisticated reasoning over multi-hop relationships and maintains the structural integrity of complex domain knowledge. GraphRAG systems can leverage graph traversal algorithms and community detection methods to identify relevant subgraphs for augmenting language model responses.



2.3 Neo4j in Knowledge Graphs

Neo4j's property graph model allows for flexible schema evolution while maintaining ACID compliance. Its Cypher query language provides intuitive graph pattern matching capabilities, making it suitable for complex transport queries involving multihop relationships. The database's native graph storage and processing engine delivers consistent performance for deep relationship traversals that would require expensive joins in relational systems. Neo4j's support for graph algorithms

and its ability to handle billions of nodes and relationships make it particularly well-suited for large-scale knowledge graph applications.

CHAPTER 3. METHODOLOGY

3.1 Architecture Overview

The system implements a three-tier architecture following established patterns for knowledge graph applications :
Tier 1: Data Integration Layer

- TfL API client with rate limiting and error handling
- JSON-to-graph transformation modules
- Batch processing capabilities for large datasets

Tier 2: Knowledge Graph Storage

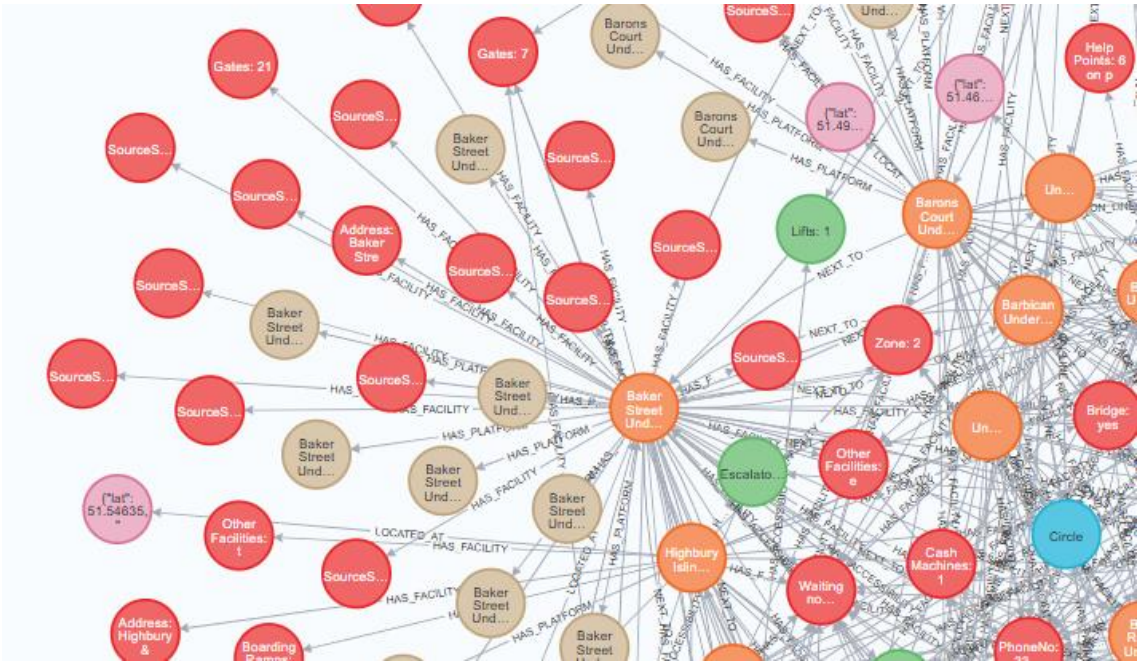
- Neo4j graph database with property graph model
- Schema constraints and indexing strategies
- Transaction management for data consistency

Tier 3: Query Interface Layer

- Natural language processing via GPT-4 API
- Cypher query generation and validation
- Result formatting and presentation

3.2 Graph Data Model

The system implements a domain-specific ontology consisting of 13 entity types and 14 relationship types. The ontological structure follows established patterns in transportation informatics.



3.2.1 Entities

Transportation Network Entities

```
Station := {id: String, name: String, lat: Float, lon:
   Float, raw: JSON}
Line := {id: String, name: String, mode: String, raw: JSON}
StopPoint := {id: String, name: String, commonName: String,
    lat: Float, lon: Float, modes: [String], raw: JSON}
Platform := {id: String, name: String, raw: JSON}
```

Operational Entities

```
Incident := {id: String, statusSeverity: Integer,
   description: String, raw: JSON}
Interchange := {id: String, stationId: String, lines: [
   String]}
Operator := {name: String, raw: JSON}
FareZone := {name: String}
```

Other Entities

```
BikePoint := {id: String, name: String, lat: Float, lon:
   Float, raw: JSON}
Timetable := {id: String, raw: JSON, nextArrivals: [Arrival
   ]}
AccessibilityFeature := {name: String}
Facility := {name: String}
```

```
Landmark := {id: String, name: String, category: String,
   lat: Float, lon: Float, raw: JSON}
Geolocation := {lat: Float, lon: Float}
```

3.2.2 Relationship Semantics

The system models relationships using directed edges with semantic labels:

Containment Relationships

`(StopPoint)-[:PART_OF]->(Station)`: Physical containment hierarchy

`(Station)-[:HAS_PLATFORM]->(Platform)`: Infrastructure composition

Network Topology

`(Station)-[:ON_LINE]->(Line)`: Service connectivity

`(Station)-[:NEXT_TO line: String]->(Station)`: Sequential positioning with line

Operational Associations

`(Station)-[:IN_ZONE]->(FareZone)`: Pricing zone membership

`(Line)-[:OPERATED_BY]->(Operator)`: Service responsibility

`(Line)-[:HAS_INCIDENT]->(Incident)`: Service disruption associations

Other associations

`(Station)-[:HAS_ACCESSIBILITY]->(AccessibilityFeature)`: Accessibility mapping

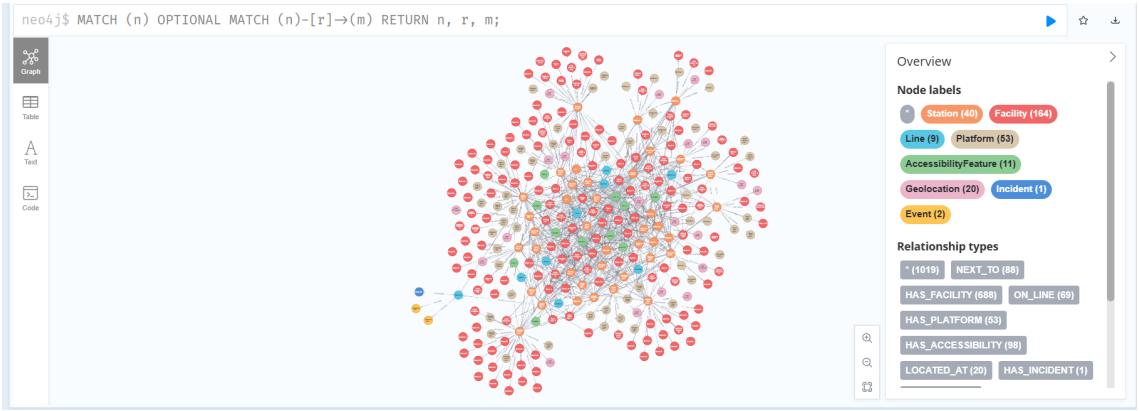
`(Station)-[:HAS_FACILITY]->(Facility)`: Facility availability

`(Station)-[:HAS_INTERCHANGE]->(Interchange)`: Interchange information

`(Station)-[:HAS_BIKEPOINT]->(BikePoint)`: Nearby bike rental points

`(Station)-[:NEAR]->(Landmark)`: Nearby landmarks

`(Station)-[:LOCATED_AT]->(Geolocation)`: Geolocation mapping



3.2.3 Schema Management

The system implements dynamic schema management through:

1. Constraint Definition: Unique constraints on primary identifiers
2. Index Creation: Performance optimization for frequent access patterns
3. Schema Introspection: Runtime discovery of available labels and relationships

```
def ensure_constraints_and_indexes():
constraints = [
    "CREATE CONSTRAINT IF NOT EXISTS FOR (n:Station)
       REQUIRE n.id IS UNIQUE",
    "CREATE CONSTRAINT IF NOT EXISTS FOR (n:Line)
       REQUIRE n.id IS UNIQUE",
    # ... additional constraints
]
```

3.3 Natural Language Query Processing

3.3.1 Query Translation Pipeline

The natural language processing component implements a multi-stage translation pipeline:

a, Schema-Aware Prompt Engineering

The system generates dynamic system prompts incorporating current database schema:

```
def translate_question_to_cypher(question: str) ->
   Tuple[str, Dict[str, Any]]:
schema = introspect_db_schema()
dynamic_schema_context = generate_schema_context(schema
   )
system_prompt = BASE_PROMPT + dynamic_schema_context
return request_gpt_translation(question, system_prompt)
```

b, Query Validation and Correction

The system implements automated query validation with correction mechanisms:

- Missing Relationship Detection:

```
        def validate_and_autofix_cypher(cypher: str,
           schema: Dict[str, set]) -> Tuple[str,
           Optional[str]]:
```

```
    missing_relationships = extract_relationships(cypher
       ) - schema["rels"]
    if missing_relationships:
        return apply_semantic_corrections(cypher,
           missing_relationships)
    return cypher, None
```

- Relationship Direction Correction: The system automatically corrects common relationship direction errors based on canonical schema patterns.

3.3.2 Error Handling and Fallback Mechanisms

a, Multi-Strategy Execution

The system implements a cascading execution strategy:

1. Primary Execution: Direct execution of GPT-generated Cypher
2. Syntactic Correction: Automatic fixing of common syntax errors
3. Semantic Fallback: Alternative query patterns for failed semantic mappings
4. Heuristic Fallback: Domain-specific heuristics for common query patterns

b, Interchange Detection Fallback

For missing `HAS_INTERCHANGE` relationships, the system implements semantic equivalence:

```
    MATCH (s:Station)-[:ON_LINE]->(l:Line)
    WITH s, collect(DISTINCT l.id) AS connectedLines
    WHERE size(connectedLines) > 1
    RETURN s.name, connectedLines
```

3.4 Data Integration and Processing

3.4.1 TfL API Integration

The system implements comprehensive TfL API integration with:
Rate Limiting: Configurable request throttling to respect API limits
Error Handling: Exponential backoff for transient failures
Data Validation: Schema validation for incoming JSON data

3.4.2 ETL Pipeline

The Extract-Transform-Load (ETL) pipeline processes TfL data in phases:

a, Entity Extraction Phase

```
    def ingest_all(radius_for_places: int = 200):
```

```
lines = tfl_get("/Line/Mode/tube")
for line in lines:
    # Phase 1: Create primary entities
    upsert_line(line)
    stop_points = tfl_get(f"/Line/{line['id']}/
       StopPoints")

    # Phase 2: Create derived entities
    for stop_point in stop_points:
        upsert_stop_point(stop_point)
        upsert_station_from_stoppoint(stop_point)
```

b, Relationship Creation Phase

```
def create_on_line_and_next_to(tx, line_id,
   stoppoint_list):
# Create sequential relationships with line context
for i, current_stop in enumerate(stoppoint_list[:-1]):
    next_stop = stoppoint_list[i + 1]
    create_sequential_relationship(current_stop,
       next_stop, line_id)
```

3.4.3 Data Quality and Consistency

The system addresses data quality challenges through:

Null Value Handling: Robust handling of missing or null API responses

Duplicate Detection: Merge operations prevent duplicate entity creation

Referential Integrity: Constraint enforcement maintains relationship consistency

CHAPTER 4. CONCLUSION AND FUTURE WORK

4.1 Conclusion

This analysis demonstrates that GraphRAG systems offer significant advantages for urban transportation information management. By combining semantic graph modeling with natural language interfaces, the system effectively addresses core challenges in data accessibility and usability. The proposed solution includes a comprehensive graph model with 13 entities and 14 relationships, capturing the full complexity of London's transportation network. Practical implementation considerations such as error handling, performance optimization, and data quality management further enhance the system's reliability. Additionally, the architecture adheres to Semantic Web principles while leveraging the performance benefits of graph databases.

From a research perspective, this work validates GraphRAG as a viable approach for domain-specific knowledge systems, especially in contexts requiring complex relationship modeling and natural language access. The integration of semantic technologies and graph reasoning provides a strong foundation for future exploration in urban informatics and intelligent transportation systems. The formalization of the ontology and the inclusion of a SPARQL endpoint open pathways for interoperability with other linked data platforms.

In terms of practical applications, the system demonstrates immediate value for transportation authorities aiming to improve public access to transit information while maintaining rich, structured data models. The approach is extensible to other urban domains such as energy, environment, and public safety, where similar complexity and data integration challenges exist. Future research should focus on scalability improvements, formal ontology development, and integration with broader urban data ecosystems to fully realize the potential of semantic transportation information systems.

4.2 Future work

- **Technical Enhancements**: The system will be extended to support multi-modal transportation networks including buses, rail, and walking paths. Real-time stream processing will be integrated to handle live updates from transport feeds. Additionally, a distributed graph architecture will be implemented using graph partitioning techniques to improve scalability and performance.
- **Semantic Web Integration**: A SPARQL endpoint will be added to enable W3C-

compliant semantic queries. RDF export capabilities will be supported in both Turtle and JSON-LD formats. The schema will be formalized using OWL-based ontologies to enhance interoperability and reasoning.

- **Advanced AI Capabilities**: The chatbot will support multi-turn conversations with context-aware dialogue management. Personalized recommendations will be enabled through user preference learning and route optimization. Predictive analytics will be introduced using machine learning models to forecast service disruptions and improve reliability.