STRATEGIC METHODOLOGY REPORT

Knowledge Graph Construction Using

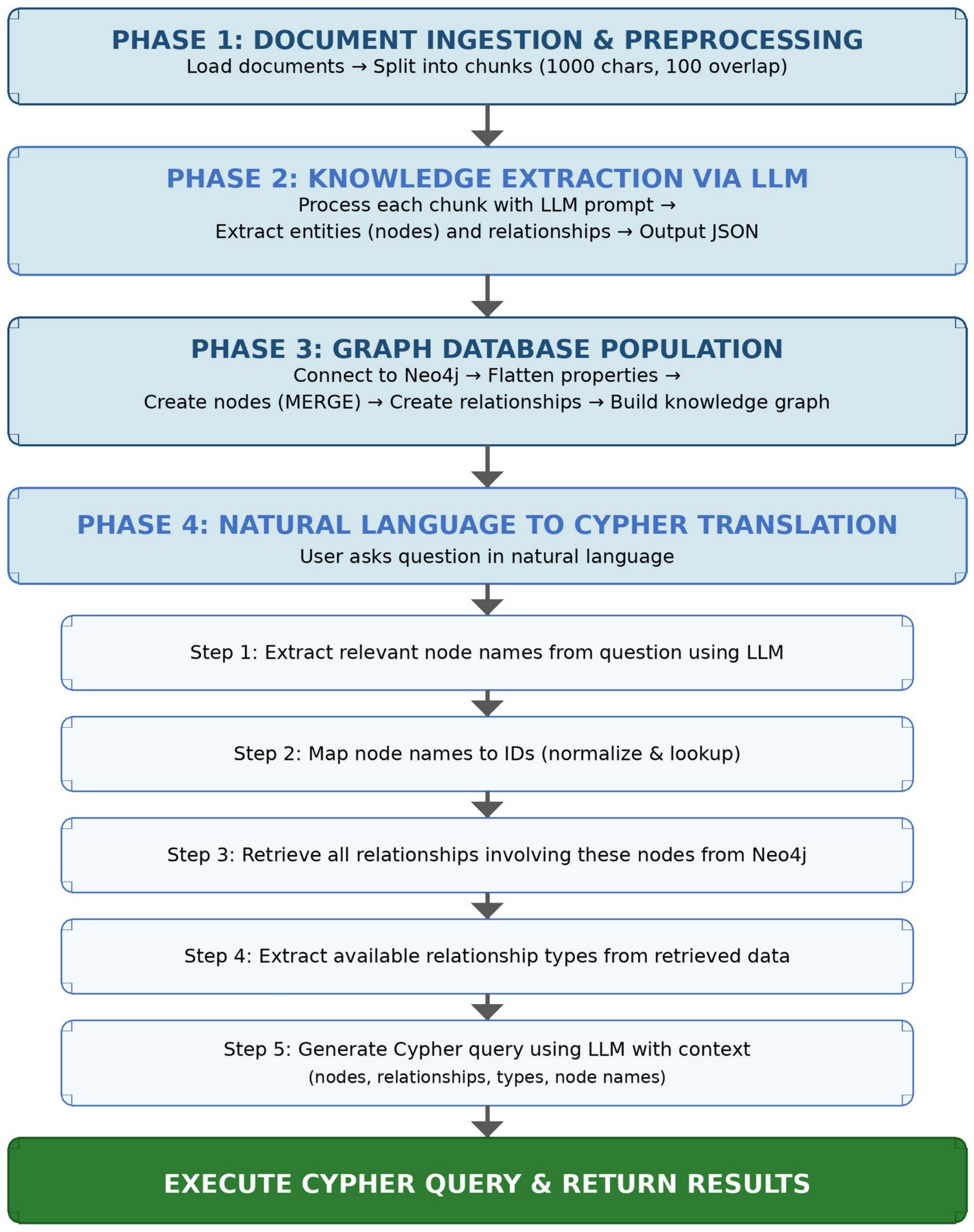
Large Language Models and Neo4j

A Comprehensive Analysis of Natural Language to Cypher Query Translation

PROCESS FLOW DIAGRAM

The following diagram illustrates the complete end-to-end workflow of the knowledge graph construction and query generation system:

KNOWLEDGE GRAPH CONSTRUCTION WORKFLOW


PHASE 1: DOCUMENT INGESTION & PREPROCESSING
Load documents → Split into chunks (1000 chars, 100 overlap)
PHASE 2: KNOWLEDGE EXTRACTION VIA LLM
Process each chunk with LLM prompt →
Extract entities (nodes) and relationships → Output JSON
PHASE 3: GRAPH DATABASE POPULATION
Connect to Neo4j → Flatten properties →
Create nodes (MERGE) → Create relationships → Build knowledge graph
PHASE 4: NATURAL LANGUAGE TO CYPHER TRANSLATION
User asks question in natural language
Step 1: Extract relevant node names from question using LLM
Step 2: Map node names to IDs (normalize & lookup)
Step 3: Retrieve all relationships involving these nodes from Neo4j
Step 4: Extract available relationship types from retrieved data
Step 5: Generate Cypher query using LLM with context
(nodes, relationships, types, node names)
EXECUTE CYPHER QUERY & RETURN RESULTS

EXECUTIVE SUMMARY

This report presents a sophisticated methodology for converting natural language queries into Neo4j Cypher queries through the strategic integration of Large Language Models (LLMs) and graph database technology. The approach demonstrates a robust, multi-layered strategy that addresses the critical challenge of bridging human communication and database query languages.

The system employs a four-stage pipeline architecture that ensures accuracy, context preservation, and intelligent query generation, making it suitable for deployment in government and enterprise environments requiring high reliability and precision.

1. INTRODUCTION

1.1 Problem Context

Knowledge graphs represent complex relationships between entities in a structured format, but accessing this information traditionally requires expertise in specialized query languages. The challenge addressed by this methodology is enabling non-technical stakeholders to query graph databases using natural language, while maintaining query accuracy and retrieving precise, relevant information.

1.2 Solution Overview

The implemented solution leverages state-of-the-art language models to transform natural language questions into valid Neo4j Cypher queries. The system incorporates document processing, intelligent text chunking, entity extraction, relationship mapping, and context-aware query generation to deliver accurate results.

2. STRATEGIC METHODOLOGY

2.1 Phase 1: Document Ingestion and Preprocessing

2.1.1 Document Loading

The system initiates by loading source documents using the LangChain DirectoryLoader with TextLoader configuration. This establishes a standardized document ingestion pipeline that handles multiple text files with UTF-8 encoding, ensuring compatibility across diverse document sources.

Key Implementation Details:

Path specification for document directory

Glob pattern matching for selective file inclusion (*.txt)

UTF-8 encoding standard for international character support

2.1.2 Intelligent Text Chunking

To optimize processing efficiency and maintain contextual coherence, the system implements a strategic text chunking mechanism using RecursiveCharacterTextSplitter. This approach divides large documents into manageable segments while preserving semantic integrity.

Chunking Parameters:

Chunk Size: 1000 characters - Optimized for LLM context window efficiency

Overlap: 100 characters - Ensures continuity across chunk boundaries and prevents loss of context at split points

Rationale: This configuration balances computational efficiency with semantic preservation, enabling the LLM to extract comprehensive entity-relationship structures without losing critical contextual information

2.2 Phase 2: Knowledge Extraction via Language Model

2.2.1 LLM Configuration and Prompt Engineering

The core intelligence of the system resides in the carefully architected prompt engineering strategy. The system employs a specialized prompt that instructs the LLM to function as an expert Knowledge Graph construction engine, specifically optimized for Neo4j integration.

Critical Prompt Components:

A. Role Definition: The LLM is explicitly positioned as a knowledge graph expert, establishing the operational context and expected output quality standards.

B. Task Specification: The system requires extraction of entities and relationships from unstructured text with strict adherence to Neo4j-compatible JSON formatting.

C. Output Constraints: Non-negotiable rules ensure structural validity and database compatibility, including mandatory JSON format, dual-key structure (nodes and relationships), and prohibition of extraneous text or markdown formatting.

2.2.2 Graph Data Model Specification

The prompt enforces a rigorous graph data model with strict structural requirements:

Node Structure:

Unique identifiers for each entity

Categorical labels (Person, Organization, Technology, Event, Location, etc.)

Property dictionaries with key-value pairs for entity attributes

Sequential ID assignment for deterministic processing

Relationship Structure:

Source and target node references

Relationship types in UPPER_SNAKE_CASE verb format

Property dictionaries for relationship metadata (dates, roles, quantities)

Mandatory four-key structure (from, to, type, properties)

2.2.3 Output Processing and Validation

The system implements robust error handling and JSON validation to ensure data integrity:

Safe JSON parsing with exception handling for malformed outputs

Markdown fence removal (```json) to clean LLM-generated formatting

Regular expression-based text cleaning for trailing commas and quote inconsistencies

Chunk-level error isolation with logging for transparency and debugging

2.3 Phase 3: Graph Database Population

2.3.1 Neo4j Connection Management

The system establishes secure, authenticated connections to the Neo4j graph database using the official Python driver. This connection layer handles authentication, session management, and transaction control.

Connection parameters include URI specification (bolt protocol), authentication credentials, and session context management for transactional consistency.

2.3.2 Data Transformation and Ingestion

The ingestion process employs sophisticated property flattening and Cypher query generation:

Property Flattening Strategy:

Recursive traversal of nested property structures

Hierarchical key generation using separator characters (underscores)

Type conversion for Neo4j compatibility (lists to comma-separated strings)

Dictionary and list handling with appropriate serialization strategies

Node Creation:

Utilizes MERGE operations to ensure idempotent node creation, preventing duplicate entities while maintaining referential integrity. The system assigns node labels dynamically and applies flattened properties directly to graph nodes.

Relationship Creation:

Employs parameterized Cypher queries with MATCH-MERGE patterns to establish connections between existing nodes. Relationship types are applied dynamically, and properties are attached to relationship instances for comprehensive metadata tracking.

2.4 Phase 4: Natural Language to Cypher Translation

2.4.1 Context-Aware Query Generation

The final and most critical phase transforms natural language questions into executable Cypher queries through intelligent context injection and LLM-based translation:

Step 1: Node Name Extraction

The system first extracts relevant entity names from the user's question by querying the LLM with the complete list of available nodes in the graph database. This step identifies which entities are pertinent to the query, ensuring that only relevant nodes are considered in the Cypher generation process.

Step 2: Node ID Mapping

Extracted node names are normalized (Unicode normalization form NFKC with whitespace removal) and mapped to their corresponding node IDs through dictionary lookups. This creates a focused set of node identifiers for relationship querying.

Step 3: Relationship Context Retrieval

Using the identified node IDs, the system executes a Cypher query to retrieve all relationships involving these nodes. This query returns the source node ID, target node ID, relationship type, and relationship properties, providing comprehensive context for query formulation.

Step 4: Available Relationship Type Analysis

The system extracts unique relationship types from the retrieved data, creating a constrained vocabulary of valid relationship types. This prevents the LLM from generating queries with non-existent relationship types.

Step 5: Cypher Query Generation via LLM

The final prompt engineering step provides the LLM with:

The user's original question

JSON representation of relevant nodes

JSON representation of relevant relationships

List of available relationship types and node names

The LLM is instructed to generate a Cypher query that answers the question using ONLY the provided nodes and relationships, outputting exclusively the query text without any additional commentary or formatting.

2.4.2 Query Execution and Result Retrieval

The generated Cypher query is executed against the Neo4j database through a managed session context. Results are returned as a list of dictionaries, enabling structured data analysis and presentation to end users.

3. TECHNICAL ARCHITECTURE

3.1 System Components

The implementation leverages a modern technology stack:

LangChain Framework: Document loading and text processing orchestration

Large Language Model: ChatOllama with GPT-OSS or equivalent models for natural language understanding and generation

Neo4j Graph Database: Production-grade graph storage with Cypher query language

Python Ecosystem: JSON processing, regex operations, and error handling utilities

Neo4j Python Driver: Official driver for database connectivity and transaction management

3.2 Data Flow Architecture

The system implements a unidirectional data flow with feedback loops:

Documents → Text Chunks → LLM Processing → JSON Entities/Relationships → Neo4j Ingestion → Knowledge Graph

User Query → Node Extraction → ID Mapping → Relationship Retrieval → Context Assembly → LLM Translation → Cypher Query → Database Execution → Results

4. KEY STRATEGIC INNOVATIONS

4.1 Context-Driven Query Generation

Unlike generic text-to-SQL systems, this methodology employs graph-aware context injection. By pre-filtering relevant nodes and relationships before LLM query generation, the system dramatically reduces hallucination and improves query precision. The LLM receives only the subset of the knowledge graph relevant to the user's question, eliminating irrelevant paths and focusing computational resources on accurate translation.

4.2 Multi-Stage Validation Pipeline

The system incorporates validation at multiple stages: JSON schema validation during extraction, node ID verification during mapping, relationship existence checking during context retrieval, and implicit Cypher syntax validation through database execution. This defense-in-depth approach ensures that errors are caught early and do not propagate through the pipeline.

4.3 Semantic Normalization

The Unicode normalization (NFKC) and whitespace stripping applied during node name matching ensures robustness against minor variations in entity naming. This allows the system to correctly identify entities even when user input includes different whitespace patterns or Unicode representations.

4.4 Chunk-Level Parallelizability

By processing text in independent chunks with overlap, the system architecture supports parallel processing for large document corpora. Each chunk can be processed independently, with the graph database handling merge operations for duplicate entities, enabling horizontal scalability.

5. BENEFITS AND APPLICATIONS

5.1 Operational Benefits

Democratization of Data Access: Non-technical stakeholders can query complex graph databases without learning Cypher syntax

Reduced Query Development Time: Questions can be posed in natural language, eliminating the iterative process of manual query refinement

Accuracy Through Context: By providing relevant graph context to the LLM, query accuracy significantly exceeds generic text-to-query systems

Scalability: The chunked processing architecture supports document collections of arbitrary size

5.2 Use Case Applications

Intelligence Analysis: Querying relationship networks from unstructured intelligence reports

Regulatory Compliance: Tracking corporate relationships and ownership structures

Academic Research: Building citation networks and collaborative research graphs

Healthcare Systems: Mapping patient-provider-facility relationships for epidemiological studies

Supply Chain Management: Analyzing supplier-product-customer networks

6. CONCLUSION

This methodology represents a significant advancement in making graph database technology accessible to non-technical users. By combining intelligent document processing, robust knowledge extraction, and context-aware query translation, the system bridges the gap between human language and database query languages.

The four-phase architecture ensures data quality, query accuracy, and system reliability, making it suitable for deployment in high-stakes government and enterprise environments. The strategic use of LLMs as both extraction engines and translation interfaces demonstrates the potential of AI-assisted database interaction.

Key achievements of this approach include:

Systematic conversion of unstructured text into structured knowledge graphs

Context-driven natural language to Cypher translation with high accuracy

Robust error handling and validation throughout the pipeline

Scalable architecture supporting large document collections

Democratization of graph database access for non-technical stakeholders

This strategic framework provides a foundation for future enhancements, including multi-hop reasoning, temporal query support, and integration with real-time data streams, positioning it as a versatile solution for complex information retrieval challenges across diverse domains.

— END OF REPORT —