Graph Database Architectures: Literary vs Medical

Overview

Both implementations use a dual-layer graph architecture combining structural hierarchy with semantic relationships extracted through NLP pipelines. One implementation focuses on ancient non-western philosophical and spiritual texts, while the other processes clinical medical records.

Literary Graph Database (Ancient Non-Western Texts)

Two-Layer Architecture

Layer 1: Document Structure

Physical organization of the corpus built from document parsing.

Hierarchy:

- Book → Chapter → Page → Paragraph

Data Source: Table of contents, document formatting, markup

Purpose: Navigate physical location of teachings

Pipeline & Libraries:

- **Document Parsing:** PyPDF2, pdfplumber, or pypdf for PDF extraction
- **Structure Recognition:** Beautiful Soup, lxml for HTML/XML parsing
- **Text Extraction:** python-docx for Word documents, pdfminer for complex PDFs
- **Hierarchy Building:** Regular expressions (re module) for detecting chapters/sections, custom parsers for table of contents
- **Metadata Extraction:** PyPDF2 for PDF metadata, python-docx for document properties
- **Graph Population:** Neo4j Python driver or py2neo to create Book/Chapter/Page/Paragraph nodes and relationships

Layer 2: Semantic Concepts

Conceptual relationships across the entire corpus extracted through NLP.

Relationships:

- Concept → RELATES_TO → Concept
- Teaching → BUILDS_ON → Teaching
- Practice → LEADS_TO → Insight

Data Source: NLP extraction from text content

Purpose: Discover interconnected wisdom patterns across all texts from multiple ancient traditions

NLP Pipeline Requirements

1. **Named Entity Recognition (NER)**
   - Extract philosophical concepts: "emptiness," "compassion," "bodhicitta," "dharma," "tao," "qi"
   - Identify teaching methods, practices, and spiritual principles
   - Custom entity dictionary for non-western philosophical terminology
   - **Libraries:** spaCy with custom trained models, Hugging Face transformers (BERT, RoBERTa), Stanford NER
2. **Dependency Parsing**
   - Extract relationships: "meditation leads to insight"
   - Subject-verb-object triples
   - Causal and conceptual relationships
   - **Libraries:** spaCy dependency parser, Stanford CoreNLP, NLTK
3. **Coreference Resolution**
   - Link pronouns to concepts: "this practice" → specific technique
   - Maintain entity continuity across paragraphs
   - Track teaching references
   - **Libraries:** NeuralCoref (spaCy extension), AllenNLP Coreference Resolution, Stanford CoreNLP
4. **Concept Normalization**
   - Map synonymous terms: "sunyata" = "emptiness" = "void", "wu wei" = "non-action"
   - Cross-tradition terminology standardization (Buddhist, Taoist, Hindu, etc.)
   - Cross-reference traditional and modern terms
   - **Libraries:** Custom dictionaries, NLTK WordNet for general synonyms, Gensim for semantic similarity

Technology Stack

- **NLP Models:** spaCy, Hugging Face transformers (BERT, RoBERTa), Stanford CoreNLP
- **Graph Database:** Neo4j
- **Vector Database:** ChromaDB (with 33% overlap)
- **Integration:** LangChain, LlamaIndex, or custom Python middleware (FastAPI, Flask)

- **Graph Drivers:** py2neo, Neo4j Python driver

Medical Graph Database (Clinical Records)

Two-Layer Architecture

Layer 1: Document Structure

Clinical record hierarchy built from EMR schema and document organization.

Hierarchy:

- Patient (MRN) → Encounter → Clinical Note → Paragraph

Data Source: EMR relational database structure, encounter IDs

Purpose: Navigate patient journey and clinical documentation

Pipeline & Libraries:

- **Database Connection:** SQLAlchemy, psycopg2 (PostgreSQL), pyodbc (SQL Server) for EMR database access
- **Data Extraction:** Pandas for querying and transforming relational data
- **HL7/FHIR Parsing:** python-hl7, fhir.resources for healthcare data standards
- **Document Parsing:** python-docx for clinical documents, PyPDF2 for scanned records
- **Text Segmentation:** NLTK sent_tokenize or spaCy sentencizer for paragraph detection
- **Graph Population:** Neo4j Python driver or py2neo to create Patient/Encounter/Note/Paragraph nodes
- **ETL Pipeline:** Apache Airflow or Prefect for orchestrating data extraction and graph updates

Layer 2: Semantic Concepts

Clinical relationships extracted from unstructured notes through medical NLP.

Relationships:

- Patient → HAS_CONDITION → Diagnosis
- Patient → PRESCRIBED → Medication
- Patient → HAS_FAMILY_HISTORY → Condition
- Medication → INTERACTS_WITH → Medication
- Patient → EXPERIENCES → Outcome

Data Source: NLP extraction from clinical notes

Purpose: Discover treatment patterns, outcomes, and complex clinical relationships

NLP Pipeline Requirements

1. **Named Entity Recognition (NER)**
   - Extract medical entities: patients, medications, conditions
   - Use clinical BERT or BioBERT models
   - Identify procedures, lab values, symptoms
   - Medical-specific entity types
   - **Libraries:** scispaCy (spaCy for biomedical text), ClinicalBERT, BioBERT, Hugging Face Med-BERT models, MedCAT (Medical Concept Annotation Tool)
2. **Dependency Parsing**
   - Extract clinical relationships: "patient takes medication"
   - Subject-verb-object triples
   - Treatment and outcome associations
   - **Libraries:** scispaCy dependency parser, spaCy, Stanford CoreNLP
3. **Coreference Resolution**
   - Link pronouns to correct entities: "he" → patient
   - Maintain patient identity across notes
   - Track medication and condition references
   - **Libraries:** NeuralCoref, AllenNLP Coreference Resolution, clinical-specific coreference models
4. **UMLS Concept Normalization**
   - Standardize medical terminology
   - Map synonyms: "heart attack" = "myocardial infarction" = "MI"
   - Link to UMLS concept identifiers
   - Prevent duplicate nodes for same condition
   - **Libraries:** QuickUMLS, MetaMap, scispaCy entity linker, UMLS API, cTAKES

Technology Stack

- **NLP Models:** scispaCy, ClinicalBERT, BioBERT, MedCAT, Hugging Face medical models
- **Graph Database:** Neo4j
- **Vector Database:** ChromaDB

- **Integration:** LangChain, LlamaIndex, or custom Python middleware (FastAPI, Flask)
- **Standards:** UMLS for concept normalization (QuickUMLS, MetaMap)
- **Graph Drivers:** py2neo, Neo4j Python driver

Pipeline Comparison

Similarities (70-80% Overlap)

Component	Both Implementations
Architecture	Dual-layer: structure + semantics
Core NLP	NER, dependency parsing, coreference resolution, normalization
Graph DB	Neo4j
Vector DB	ChromaDB
Integration	Python orchestration layer
Workflow	Extract → Normalize → Graph construction

Differences

Aspect	Literary	Medical
Domain Models	General language models	Clinical BERT, BioBERT
Entity Types	Philosophical concepts, practices	Patients, conditions, medications
Normalization	Cross-tradition terminology mapping	UMLS standardization
Structure Source	Document markup, TOC	EMR schema, encounter IDs
Complexity	Custom dictionary needed	Established medical ontologies

---

Integration Layer

Purpose

Coordinates queries across relational database (EMR), vector database (text search), and graph database (relationships).

Components

1. **Query Orchestration**
   - Routes queries to appropriate database
   - Combines results from multiple sources
2. **Common Approaches**

- LangChain framework for chaining operations
- Custom Python API middleware
- FastAPI or Flask service layer

3. **Workflow Example**
   - User query → Vector search finds relevant clinical notes
   - Graph traversal reveals patient relationships and context
   - Relational DB validates structured data constraints
   - Results combined and returned to user

Use Case Examples

Literary Application

Query: "Find all teachings about emptiness that relate to compassion"

- **Vector DB:** Finds paragraphs discussing emptiness
- **Graph Layer 2:** Traverses semantic relationships to compassion concepts
- **Graph Layer 1:** Shows which books and chapters contain these teachings

Medical Application

Query: "Find diabetic patients on metformin with family history of hypertension"

- **Relational DB:** Filters patients with diabetes diagnosis
- **Vector DB:** Searches clinical notes for metformin mentions
- **Graph Layer 2:** Traverses family history relationships
- **Graph Layer 1:** Navigates to relevant encounter notes

Key Insight

Both implementations are **structurally identical** in architecture and pipeline design. The primary differences are domain-specific models and terminology. Building the literary graph database first provides a fully reusable framework for the medical implementation—only the NLP models and entity dictionaries need to be swapped out.