LLM-Driven Knowledge Graph Construction in Biomedicine

Introduction

The biomedical domain presents unique challenges for building knowledge graphs (KGs). Textual sources such as electronic medical records (EMRs), journal articles and clinical notes contain rich information about diseases, treatments and interactions, but the language is noisy, full of domain-specific abbreviations and continuously evolving. Large language models (LLMs) have recently shown promise for extracting and structuring information from unstructured text. In particular, Arsenyan *et al.* proposed an **end-to-end method** that uses LLMs to construct KGs from EMR notes. Their approach aligns disease mentions using the Unified Medical Language System (UMLS), applies BioBERT for named entity recognition, and then uses various LLMs to extract relations via guided prompts 【376494866725231†L119-L127】 【376494866725231†L248-L267】. A more recent survey explores the synergy between KGs and LLMs in biomedicine: knowledge graphs store accurate, explicit and modifiable facts, whereas LLMs provide flexible language understanding 【402605036019548†L122-L155】. The combination of the two aims to overcome LLM hallucinations and data scarcity 【402605036019548†L152-L164】.

This report designs a modular project for building a biomedical KG using LLMs. It synthesises insights from the literature and translates them into a practical architecture that supports both local prototyping and scalable deployments.

Goals and Use Cases

The envisioned system should:

1. **Extract entities and relations from unstructured biomedical text.** The pipeline must handle noisy EMR notes, abbreviations and varying terminologies. It should also be adaptable to scientific literature when scaling up.
2. **Construct an interoperable knowledge graph.** The KG should integrate with existing biomedical ontologies such as UMLS, MeSH and Gene Ontology. Using common identifiers reduces redundancy and enables downstream analytics.
3. **Support search and question answering.** A primary use case is retrieval-augmented question answering (RAG), where the KG and vector embeddings provide context to an LLM. Reasoning and multi-hop traversal are desirable but deferred to future iterations.
4. **Enable human-in-the-loop validation.** Domain experts must be able to review and correct extracted triples. Semi-automated evaluation can use gold-standard datasets or existing KGs for comparison 【790445754267283†L29-L38】.
5. **Scale from prototypes to millions of documents.** Early development will target small corpora (~hundreds of documents) before scaling to millions (e.g. full PubMed). The architecture should allow distributed storage and compute for embedding and graph storage.

Architectural Overview

Figure 1 outlines the proposed modular architecture. Each numbered component can be developed and tested independently.

1. **Data Ingestion.** The system must ingest text from multiple sources: EMR notes, PubMed abstracts, clinical trial registries and more. An ingestion module handles de-duplication, language detection, document segmentation and metadata extraction. For EMR notes, UMLS can be used up front to identify disease terms and variations 【376494866725231†L248-L267】.
2. **Pre-processing and Text Splitting.** Documents are split into overlapping chunks to create retrieval units. Arsenyan *et al.* found that providing additional context beyond the immediate prompt helps LLMs avoid bias and extract relations correctly 【376494866725231†L130-L170】. The chunk size and overlap should be configurable.
3. **Embedding and Vector Store.** Each chunk is embedded into a dense vector using a cloud LLM embedding (e.g. text-embedding-ada-002) or a local model. Embeddings are stored in a vector database (FAISS for prototyping, Qdrant or Pinecone for scaling). This supports efficient similarity search.
4. **Retrieval.** Given a query or extraction task, the system retrieves relevant chunks from the vector store. This step forms the basis of retrieval-augmented generation.
5. **Named Entity Recognition and Linking.** Before relation extraction, biomedical named entities (diseases, drugs, genes, symptoms) must be recognised and linked to canonical identifiers. Pre-trained models like BioBERT and SciSpacy provide accurate NER, while UMLS, MeSH and GO enable entity alignment and disambiguation.
6. **Relation Extraction using LLMs.** Retrieved chunks and recognised entities are fed to an LLM using a guided prompt. Arsenyan *et al.* demonstrated that instruction-based prompting combined with retrieval context improves structured outputs 【376494866725231†L341-L353】. The LLM returns subject–predicate–object triples in JSON format. Prompt engineering functions live in `prompts.py`.
7. **Post-processing and Triple Normalisation.** Extracted triples are validated to remove duplicates, map synonyms to canonical IDs, and filter out improbable relations. Human validators can review candidate triples through a curation interface.
8. **Graph Construction and Storage.** Validated triples are inserted into a graph database (Neo4j, Neptune). The initial prototype uses NetworkX for simplicity but exposes an abstraction for graph operations.
9. **Query and Search.** Users can perform graph queries (e.g. “find all treatments for a disease”) or natural language queries. For the latter, the system performs RAG: it retrieves relevant triples and embeddings, then uses an LLM to generate the answer using only the provided context.
10. **Evaluation and Feedback.** Generated KGs are evaluated against manually curated ground truth using precision, recall, F1 score, graph edit distance and semantic similarity metrics 【790445754267283†L29-L38】. Feedback mechanisms allow continuous improvement.

Deployment and Scaling

Local Development

During early development, the entire pipeline can run on a laptop. Text documents and graphs reside in memory. The only external dependency is the OpenAI API for embeddings and relation extraction. This mode allows rapid iteration on prompts and extraction logic.

Cloud and Distributed Deployment

To process millions of documents and build a large KG, several components must scale:

- **Embedding.** Partition documents and process them in parallel across worker nodes. Use persistent vector stores (Qdrant, Pinecone) to keep embeddings. Avoid using GPUs for PyTorch models initially; the POP llmclient.py can route requests to cloud providers.
- **Graph Storage.** Deploy a graph database cluster to store billions of triples. Neo4j supports clustering and can run on cloud services. Each worker writes its triples to the database through an API.
- **Real-Time Ingestion.** For continuous updates (e.g. PubMed streaming), set up a message queue (Kafka) and microservices for ingestion, pre-processing, embedding and extraction. The modular design allows these services to be scaled independently.

Human-in-the-Loop Validation

LLM outputs are prone to hallucination and errors. Arsenyan *et al.* note that decoder-only models require guided prompts to produce structured outputs and still need careful validation 【376494866725231†L116-L127】. A curation interface should present extracted triples to domain experts, who can accept, reject or edit them. Their feedback can be fed back into fine-tuning or prompt tuning. Automatic checks (e.g. verifying that a drug–disease relationship exists in UMLS) can filter obvious mistakes.

Evaluation and Benchmarking

Robust evaluation is essential before deploying a biomedical KG. Bhatt *et al.* compare GPT-4, LLaMA 2 and BERT for KG generation and evaluate their outputs using precision, recall, F1 score, graph edit distance and semantic similarity 【790445754267283†L29-L38】. A similar evaluation suite should be implemented. Start with small, manually annotated corpora to compute metrics. Later, cross-validate against existing KGs such as SemRep or BioKG.

Conclusion

Large language models offer a powerful new tool for biomedical knowledge extraction, but their use must be tempered with careful prompt design, ontological grounding and human oversight. The modular architecture presented here lays a foundation for building a scalable LLM-driven knowledge graph. By integrating best practices from recent research—such as guided prompting and canonical entity alignment 【376494866725231†L119-L127】

【376494866725231†L248-L267】—the project aims to construct accurate, up-to-date biomedical KGs that support search, question answering and future reasoning applications.